AD-A264 230

JMENTATION PAGE

(2)

2 REPORT DATE	3 REPORT TYPE AND DATES COVERED
	FINAL/01 OCT 90 TO 31 OCT 92

4 TITLE AND SUBTITLE
MULTIPROCESSING SYSTEMS: RELIABILITY MODELLING & ANALYSIS USING MULTIMODE COMPONENTS & DEPENDENT FAILURES (U)

5 FUNDING NUMBERS
2304/ES
AFOSR-91-0025

6 AUTHOR(S)
Professor Suresh Rai

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Louisiana State University
Electrical/Computer Engineering
Baton Rouge LA 70803

8 PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 93 0251

9 SPONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
110 DUNCAN AVE, SUTE B115
BOLLING AFB DC 20332-0001

10. SPONSORING MONITORING AGENCY REPORT NUMBER
AFOSR-91-0025

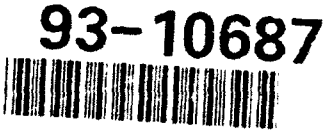
11. SUPPLEMENTARY NOTES

12a DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The researchers have established simple and efficient algorithms for terminal reliability (TR) and broadcast reliability (BR) evaluation of the shuffle-exchange network with an extra stage (SENE). In the SENE, each input is connected to each output by a pair of complete binary trees such that the input is connected by a directed edge to each of the roots, ad the leaves of both trees are identical. These very regular paths from an input to the outputs offer the structure necessary to solve the TR BR problems efficiently.

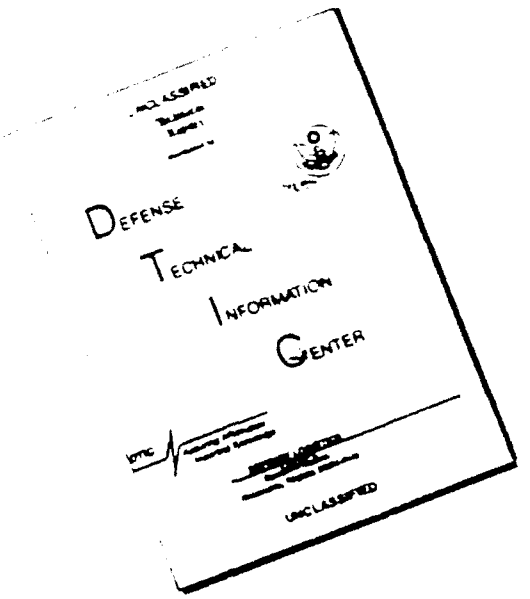


14. SUBJECT TERMS	15 NUMBER OF PAGES 105
	16 PRICE CODE

17 SECURITY CLASSIFICATION OF REPORT	18 SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20 LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR(SAME AS REPORT

Illegal width.

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Air Force Office of Scientific Research grant AFOSR-91-0025
Grant duration: October 1, 1990 to October 31, 1992
Report covering the period: October 1, 1990 to October 31, 1992

Technical Report to the Air Force Office of Scientific Research

Multiprocessing Systems: Reliability Modelling and Analysis Using Multimode Components and Dependent Failures

Suresh Rai and Jerry L. Trahan

February 1992

Final Technical Report
Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA 70803

Telephone and e-mail:
Suresh Rai — (504) 388-4832, suresh@max.ee.lsu.edu
Jerry L. Trahan — (504) 388-4830, trahan@max.ee.lsu.edu

PART A

Air Force Office of Scientific Research grant AFOSR-91-0025
Grant duration: October 1, 1990 to September 30, 1992
Report covering the period: October 1, 1990 to September 30, 1991

Report to the Air Force Office of Scientific Research

Multiprocessing Systems: Reliability Modelling and Analysis Using Multimode Components and Dependent Failures

Suresh Rai and Jerry L. Trahan

October 29, 1991

Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA 70803

Telephone and e-mail:
Suresh Rai — (504) 388-4832, suresh@max.ee.lsu.edu
Jerry L. Trahan — (504) 388-4830, trahan@max.ee.lsu.edu

Introduction

With the advancement of research and development in multiprocessing systems, researchers are focusing greater attention on the design of the interconnections between components of a system. Reliability analysis and performance evaluation are the essential aspects of any study into the effectiveness of a structure. We have begun investigating the reliability problem for two types of multiprocessing interconnection schemes: multistage interconnection networks and hypercubes. A group comprising Suresh Rai, Jerry L. Trahan, and three students: T. Smailus, S. Ananthakrishnan, and P. Paragi, was formed to work on a range of subtopics. Besides these, a Ph.D. student of Dr. Rai, S. Soh, and two M.S. students of Dr. Trahan, A. Kulkarni and R. Ahmed, looked into some of the related aspects of the problem. Dr. Rai also interacted with Dr. S. Latifi of the University of Nevada at Las Vegas to devise a strategy for bounding hypercube reliability. A bibliography at the end of this report compiles our research results obtained so far on the topic.

Results

MULTISTAGE INTERCONNECTION NETWORKS

We have established simple and efficient algorithms for terminal reliability (TR) and broadcast reliability (BR) evaluation of the shuffle-exchange network with an extra stage (SENE) [1-3]. In the SENE, each input is connected to each output by a pair of complete binary trees such that the input is connected by a directed edge to each of the roots, and the leaves of both trees are identical. These very regular paths from an input to the outputs offer us the structure necessary to solve the TR and BR problems efficiently. We first developed a sum of disjoint products approach to this problem [2]. Later, we developed an efficient algorithm for BR evaluation of an $N \times N$ SENE by a recursive approach, resulting in a recurrence equation that can be evaluated within a constant amount of time for each of $\log N$ levels of recursion [1]. This result establishes that the problem of evaluating the broadcast reliability for a SENE is not only not NP-hard, as is the case for a general network, but has a very efficient algorithm. We extended this algorithm to an efficient algorithm for the K-terminal reliability problem.

If we consider a deterministic model for a network in which each component is given as working or failed, then we can study a set of decision problems analogous to the reliability problems of interest in the stochastic model. For example, a terminal decision problem is the problem of determining whether a path exists from a specific source to a specific terminal, given a network with a known set of failures. Efficient algorithms are of interest to determine whether a given network with certain working and failed components at a certain point in time can effect a

needed set of connections. These algorithms are also of interest as they may provide techniques useful for the reliability evaluation of specific networks.

For MINs, we have developed a set of approaches for the terminal decision, broadcast decision, and network decision problems, and for the general S, T decision problem for an input set S and output set T [1]. These approaches are based on either testing for the existence of an appropriate pathset or testing for the nonexistence of a cutset. For the broadcast and network decision problems, the cutset approach leads to better algorithms as we can more concisely describe and locate cutsets than pathsets.

HYPERCUBES

The decision problem for hypercubes is explored considering a deterministic model for the system. A real world problem is modelled assuming a given set of failures in a cube, which may be restricted to subcube (a 0-subcube represents a node) failures only, or link failures only, or both subcube and link failures. The failures could be of a permanent nature or of a temporary nature. A permanent failure type refers to a complete outage scenario. A temporary fault is nothing but unavailability of an i-subcube which is currently busy with some processes and is involved in executing an algorithm. The question is then asked how to determine the size and location of the maximal dimension available (fault-free) subcube. To help answer this problem we have defined two operators, namely, # and $. We used these operators to develop a method for identify all maximal size, fault-free subcubes contained in a faulty cube [3]. (See attached report for details.)

Additionally, we have addressed the problem of dynamically allocating subcubes of a hypercube to multiple tasks [4]. Our allocation algorithm falls into the category of available cube techniques which offer the advantage of quickly recognizing whether or not a requested subcube is available in the free list of subcubes. The allocation is done using a best-fit concept to select a subcube for allocation, which in turn utilizes the notion of overlap-syndrome to quantify the overlap among free subcubes. Our technique has full subcube recognition ability and thus recognizes more subcubes as compared to bit mapped techniques: buddy, gray code and its variants. We have also developed a corresponding deallocation algorithm. The algorithms work with the previous method for handling faulty nodes and links in the hypercube.

A probabilistic model for hypercubes is considered in [5, 6]. The studies are confined to terminal and network reliability evaluations for their exact and approximate expressions.

GENERAL

In addition, we have developed an efficient Boolean approach, called CAREL [7], to solve reliability problems in general networks. The effect of preprocessing of path or cut terms on the overall reliability expression is experimentally determined in [8]. Moreover, a capacity related

reliability problem where two parameters such as availability of a link and its capacity are used to quantify the reliability measure [9]. All these efforts are helpful in understanding different aspects of the reliability evaluation problem.

Ongoing Efforts

We are currently pursuing the following directions in our work.

For MINs, the standard reliability analysis model assumes that only switches can fail, that links are perfectly reliability, that failures are statistically independent, and that a switch is either completely working or completely failed. Such restrictive assumptions are standard for reliability analysis problems in general networks that are already intractable even with these assumptions. As we have developed efficient algorithms for reliability analysis of MINs, we are seeking to loosen these unrealistic restrictions on the analysis [10]. Consequently, we are developing methods to incorporate link failures, dependence between component failures, and multimode components into the analysis. Currently, we are handling each assumption separately, but our intention is to develop reliability evaluation methods incorporating all these more realistic assumptions. We are also working to develop a network reliability evaluation algorithm for the SENE.

For hypercubes, our efforts are in two directions. First, we are working to improve our subcube allocation and deallocation schemes. Second, we are studying reliability evaluation of the hypercube, working from the terminal reliability results obtained as stated above.

Bibliography

[1] J. L. Trahan and S. Rai (1991), "Reliability Evaluation and Decision Problems in Extra Stage Shuffle-Exchange MINs," submitted to *Networks*.

[2] A. Kulkarni and J. L. Trahan (1991), "Broadcast Reliability Evaluation of Multistage Interconnection Networks," *Proc. Southeastcon '91*, Williamsburg, VA.

[3] S. Rai and J. L. Trahan (1991), "ATARIC: An Algebraic Technique to Analyse Reconfiguration for Fault Tolerance in a Hypercube," to appear *Proc. Symp. Par. and Distr. Processing*.

[4] S. Rai, J. L. Trahan, and T. Smailus (1991), "Processor Allocation in Faulty Hypercube Multiprocessors," submitted to 6th International Parallel Processing Symposium.

[5] S. Latifi and S. Rai (1991), "A Robustness Measure for Hypercube Multiprocessors," submitted to *Computer Networks and ISDN*.

[6] R. M. Ahmed and J. L. Trahan (1991), "Two-Terminal Reliability of Hypercubes," *Proc. Southeastcon '91*, Williamsburg, VA.

[7] S. Soh and S. Rai (1991), "CAREL: Computer Aided Reliability Evaluator for Distributed Computer Networks," *IEEE Trans. Par. Distr. Sys.*, vol. 2, no. 2, pp. 199-213.

[8] S. Rai and S. Soh (1991), "A Computer Approach for Reliability Evaluation of Telecommunication Networks with Heterogeneous Link-Capacities," *IEEE Trans. Reliability*, vol. 40, no. 2, pp. 441-451.

[9] S. Soh and S. Rai (1991), "Experimental Results on Preprocessing of Path/Cut Terms in the Sum-of-Disjoint-Products Technique," *Proc. INFOCOM'91*, pp. 533-542.

[10] S. Ananthakrishnan (1991), "Reliability Evaluation of Multistage Interconnection Networks with Multimode Failures," M.S. thesis, Dept. of Electrical and Computer Engineering, Louisiana State University.

ATARIC: An Algebraic Technique to Analyse Reconfiguration for Fault Tolerance in a Hypercube

Suresh Rai and Jerry L. Trahan

Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA 70803

Abstract

The hypercube architecture is a popular topology for many parallel processing applications. Several researchers have analysed the performance and dependability aspects of this architecture or its variants. Fault tolerance by reconfiguration is another important problem in a large distributed computing environment, for continued operation of the hypercube multiprocessors after the failure of one or more i-subcubes and/or links. This paper considers the fault tolerance issue and presents an algebraic technique, called ATARIC, to analyse the problem. ATARIC (Algebraic Technique to Analyse Reconfiguration for fault tolerance In a hyperCube) uses algebraic operators to identify the maximum dimensional fault-free subcube, and it thus helps in achieving graceful degradation of the system. We analyse the complexity of our algorithm. ATARIC is efficient as compared to the algorithm of Özgüner and Aykanat [4], where the inclusion-exclusion principle is used. Examples illustrate the approach.

1. Introduction

Hypercube multiprocessors have been the focus of many researchers over the past few years. The appealing properties of the hypercube such as node and edge symmetry, logarithmic diameter, high fault resilience, scalability, and the ability to host popular interconnection networks, viz., ring, torus, tree, and linear array, have made this topology an excellent candidate for many parallel processing applications [1-3]. Conceptually, the hypercube interconnection network is a multidimensional binary cube with a processing element (PE) at each of its nodes. An 'n' dimensional hypercube, B_n, has 2^n processors and $n2^{n-1}$ links. Each processor has its own local memory and interprocessor communication is done by explicit message passing.

Several variants to the hypercube topology such as cube-connected-cycles (CCC), generalized hypercube (GHC), bridged hypercube (BHC), twisted hypercube (THC), folded hypercube (FHC), and star graph [2] are described in the literature.

The suitability of a parallel architecture is evaluated by analyzing its performance and reliability aspects. Several researchers have investigated hypercube systems using performance metrics such as number of nodes and links, connectivity, diameter, average distance, cost, expandability, etc. [3] with or without faults in B_n. Few researchers have paid attention to dependability issues. Dependability (in terms of reliability or availability) prediction of a hypercube architecture uses a stochastic graph model of B_n. Note that this prediction is quite essential since hypercubes have the potential of use in critical applications. Reliability (availability) prediction is important for systems with short (long) mission times.

Fault tolerance by reconfiguration is another important problem in a large distributed computing environment, for continued operation of the hypercube multiprocessor after the failure of one or more i-subcubes (a 0-subcube is a node or PE) and/or links. Algorithms for diagnosing faulty processors and links have been given [5-7]. Once the faulty elements have been identified, graceful degradation can be achieved by

This work was supported in part by the Air Force Office of Scientific Research under grant AFOFR-91-0025.
Authors' e-mail addresses: Suresh Rai: suresh@max.ee.lsu.edu; Jerry L. Trahan: trahan@max.ee.lsu.edu

To appear: Proc. Symp. Par. and Distr. Processing

reconfiguring the multiprocessor and the distributed algorithm running on the multiprocessor [4]. Fortunately, most parallel algorithms can be formulated with the dimension n of the hypercube being a parameter of the algorithm [8]. Hence, the reconfiguration problem in a hypercube multiprocessor reduces to finding the maximum dimensional fault-free subcube(s). A subcube is a subset of a hypercube which preserves the properties of the hypercube.

References [4,8] provide simple procedures to find the maximum dimension "d" of a fault-free subcube. However, as indicated by Özgüner and Aykanat [4], the procedure of Becker and Simon [8] does not always find the maximum dimension and, furthermore, does not construct the set of fault-free d-subcubes. Özgüner and Aykanat [4] made use of the principle of inclusion-exclusion in algorithms that always find d and also the number of fault-free d-subcubes or the complete set of fault-free d-subcubes.

This paper introduces a new algebraic technique to analyse reconfiguration for fault tolerance in a hypercube, henceforth called as ATARIC. The ATARIC addresses the dependability problem also. But, it is different from stochastic graph model based dependability measures such as K-terminal reliability, subcube reliability, and task-based reliability presented in the literature [9-11]. We present two operators, namely # (sharp) and $ (dollar), to help describe ATARIC. The # operator is quite general and a modification to it finds use in PLA testing [12] and reliability computation of general networks [13,14,19]. Similar to the algorithm of Özgüner and Aykanat [4], the proposed technique is also formulated to run on a single processor which would typically be the host or the resource manager in a commercial hypercube system.

The layout of the paper is as follows. Section 2 provides a discussion on the hypercube and its properties. The ATARIC operators, # and $, are presented in Section 3. Section 4 gives the algorithm and illustrates the technique with examples. The complexity issues described in Section 5 show that our method is more efficient than the previous approach [4]. Finally, Section 6 concludes the paper. An appendix provides a proof of correctness of the algorithm.

2. Preliminaries

2.1. Hypercube concepts

An n-dimensional hypercube is defined as $B_n = K_2 \times B_{n-1}$, where K_2 is the complete graph with two nodes, B_0 is a trivial graph with one node and $\times$ is the product operation on two graphs [16]. Let B_n be modeled as a graph G(V,E) with $|V| = 2^n$ and $|E| = n\,2^{n-1}$. The graph G(V,E) is both node and link symmetric. Each node in G(V,E) represents a processor and each edge represents a link between a pair of processors. Nodes are assigned binary numbers from 0 to $(2^n - 1)$ such that addresses of any two neighbors differ in only one bit position. Using an n-tuple, a PE in B_n is represented by $(b_{n-1},\ldots,b_i,\ldots,b_0)$, where $b_i \in \{0,1\}$. Two adjacent nodes which differ in the i-th bit are said to be in direction i ($0 \le i \le n-1$) with respect to each other. A subcube in a hypercube B_n is a subset of a hypercube which preserves the properties of a hypercube. It is represented by an n-tuple $\{0,1,x\}^n$. Coordinate values "0" and "1" can be referred to as fixed or bound coordinates and "x" as free. An i-dimensional cube (or i-subcube) in B_n has (n-i) bound coordinates and i free coordinates. Hence, there are $2^{n-i}\binom{n}{n-i}$ different possible i-subcubes in B_n. We will use the terms node and 0-subcube interchangeably throughout the paper since they denote the same object. For links, we introduce a different notation to differentiate between a link and a 1-subcube. Here, a link has coordinate values as: 0, 1, and q. For example, 100q denotes the link with end nodes 1000 and 1001 in Figure 1. An n-tuple describing a link contains exactly one q. The position of the coordinate q in one of the n coordinate positions indicates the adjacency direction for the end nodes. The reader is suggested to refer to [16-18] for other interesting properties of a hypercube graph.

2.2. Fault models

The reconfiguration problem for hypercubes is explored considering the faults located at i-subcubes and/or links. In Figure 1, when a 2-subcube xx10 is faulty, we assume that all the four nodes forming the 2-subcube along with their interconnecting links are unavailable. For i=1, a 1-subcube consists of a link and its two end nodes. Thus, a faulty 1-subcube assumes the entire group consisting of the link and its two end nodes are unavailable. Node failure is considered as a special case of i-subcube fault, where i = 0. We assume that a node and all edges connected to that node are removed from the graph.

A link failure has the effect of deleting the particular link from G(V,E). We consider a link failure to be total, i.e., we do not assume the case where a full duplex link fails in one direction but functions in the other direction.

This assumption allows the use of an undirected graph as a network model as opposed to a directed graph [5].

Note that a link and/or node may be faulty due to the presence of some hardware failure in the system. When some task is currently being executed on an i-subcube, the said i-subcube is temporarily unavailable and may be considered as faulty from the viewpoint of reconfiguring the multiprocessor to run an additional task.

Let f_1= 0000 and f_2= 0100 be two faulty nodes in B_4 shown in Figure 1. The faulty processor f_2 belongs to the 2-subcubes xx00, x1x0, 0xx0, x10x, 0x0x, 01xx and, therefore, destroys these subcubes. The total number of i-subcubes destroyed by a faulty processor is $\binom{n}{n-i}$ [4]. Note that the set of i-subcubes destroyed by a number of faulty PEs may not be disjoint. For example, 0x0x, xx00, and 0xx0 are also covered by the fault f_1. In what follows, we describe the coordinate # and S operations and discuss an efficient technique to locate the maximum dimension (fault-free) subcube using these operators.

3. ATARIC operators

The # and S operators, defined below, are used to find a non-faulty maximum dimension subcube in the presence of subcube and link failures. To give an algebraic definition we first define coordinate # and S operations as given in Table 1. Let c_r and f_s be two cubes of length n such that

$c_r = (a_{n-1},\ldots,a_i,\ldots,a_0)$ and

$f_s = (b_{n-1},\ldots,b_i,\ldots,b_0)$,

where $a_i \in \{0,1,x\}$ and $b_i \in \{0,1,x\}$ ($b_i \in \{0,1,q\}$) where the fault type is a subcube (link) failure. The # operation between c_r and f_s is defined by:

$$c_r \# f_s = \begin{cases} c_r & \text{; if } a_i \# b_i = y \text{ for any } i \\ \varnothing & \text{; if } a_i \# b_i = z \text{ for all } i \\ \bigcup_{i \in P} a_{n-1}\ldots a_{i+1}\alpha_i a_{i-1}\ldots a_0 & \text{; otherwise,} \end{cases} \quad (1)$$

where $P = \{i \mid a_i \# b_i = \alpha_i = 0 \text{ or } 1\}$

If C is a set of cubes, $C = \{c_1,\ldots,c_h\}$, then define $C \# f_s = \bigcup_{r=1}^{h} c_r \# f_s$. The sharp (#) operator is introduced by Miller [15], and its application to PLA testing and reliability analysis in general networks is described in [12-14].

The S operator is similar to that of Equation (1). Table 1 illustrates the coordinate S operator. Let $c_r \$ f_s = c_r$, if $a_i \$ b_i = y$ for any i; else, let $c_r \$ f_s = X \cup Y \cup Z$, where for some j, $a_j \$ b_j = t$, and

$$X = \begin{cases} \varnothing; \text{if } a_j \$ b_j = t \text{ for some } j \text{ and } a_i \$ b_i = t \text{ for all } i \neq j, \\ \bigcup_{i \in P} a_{n-1}\ldots a_{i+1}\alpha_i a_{i-1}\ldots a_0; \text{ otherwise, where} \\ \qquad P = \{i \mid a_i \$ b_i = \alpha_i = 0 \text{ or } 1\}. \end{cases}$$

$Y = (a_{n-1},\ldots,a_{j+1},0,a_{j-1},\ldots,a_0)$, and

$Z = (a_{n-1},\ldots,a_{j+1},1,a_{j-1},\ldots,a_0)$. (2)

Note, both # and S operators are non-commutative (i.e., $a_i \circ b_i \neq b_i \circ a_i$, where "o" may describe # or S operators). The following properties of the "o" operator follow immediately from the definition.

1) $c_r \circ f_s = c_r$; if $c_r \cap f_s = \varnothing$
2) $c_r \circ f_s \subseteq c_r$
3) $c_r \circ f_s \neq f_s \circ c_r$
4) $(c_r \circ f_s) \circ f_m \neq c_r \circ (f_s \circ f_m)$; non-associative
5) $(c_r \cup f_s) \circ f_m = (c_r \circ f_m) \cup (f_s \circ f_m)$
6) $(c_r \cap f_s) \circ f_m = (c_r \circ f_m) \cap (f_s \circ f_m)$

From 5) and 6) it is obvious that the operators satisfy the distributive law over the $\cup$ and $\cap$ operations. Moreover, the following interesting property is also possessed by these operators.

7) $(c_r \circ f_s) \circ f_m = (c_r \circ f_m) \circ f_s$

Miller [15] provides interesting reading material for some of these properties.

Example 1. Let 0100 be a faulty processing element in B_4. Using c_1 = xxxx (i.e., B_4 is assumed to be non-faulty initially) we have -

```
c1 # f1 =xxxx
         0100
         ----
         1011        ; using coordinate # operation.
```

From Equation (1) we get -

$c_1 \# f_1 = \{1xxx, x0xx, xx1x, xxx1\}$.

Example 2. In B_5, let xx10x describe a subcube unavailability. Assuming c_1 = xxxxx, we get -

```
c1 # f1 = xxxxx
          xx10x
          -----
          zz01z      ; using coordinate # operation.
```

From Equation (1) we get -

$c_1 \# f_1 = \{xx0xx, xxx1x\}$.

Note that $c_1 \# f_1 = \{xx0xx, xxx1x\}$ provides the information about working or available subcubes.

Example 3. Consider a faulty link 00q (= f_1) in B_3. For c_1 = xxx, we have

```
c1 $ f1 = xxx
          00q
          ---
          11t      ; using coordinate S operation.
```

From Equations (1) and (2) we get -

c_1 \$ f_1 = {1xx, x1x, xx1, xx0}.

The first two values are obtained for t = z and the next two values are for t = 1 or 0. Thus, the fault-free 2-subcubes are: 1xx, x1x, xx1, xx0.

4. Algorithm

The operators discussed in Section 3 are useful in the understanding of ATARIC. The steps of the algorithm are as follows.

Step 0. [Given] Fault list $f_1, f_2 \ldots f_m$. An element f_i describes a subcube and/or link failure and uses the representation described earlier.

Step 1. [Initialize] $C_1 = \{c_1\}$, where c_1 = x x x ... x. The cube c_1 is an n-tuple and assumes that B_n is non-faulty (available) initially. Set i = 1.

Step 2. Compute $C_{i+1} = C_i$ o f_i, (3)
where the operation "o" is either # or \$ depending on f_i representing a subcube or link failure, respectively. Note that C_{i+1} may have one or more than one subcubes.
If $C_{i+1} = \emptyset$, go to step 4.

Step 3. Increment i = i+1.
Check i > m
a) if yes, go to step 4.
b) if no, go to step 2.

Step 4. Stop.

Illustrating Example. Let the faulty elements in a 6-hypercube B_6 be

f_1 = 011000 f_2 = 000101 f_3 = 001q10
f_4 = 100010 f_5 = 110111 f_6 = 101101

Note f_1, f_2, f_4, f_5, and f_6 describe faulty nodes while f_3 describes a faulty link. Initializing c_1 to x x x x x x and following the steps of the algorithm, we have

C_1 # f_1 = {1xxxxx, x0xxxx, xx0xxx, xxx1xx, xxxx1x, xxxxx1}

C_2 # f_2 = {1xxxxx, xxxx1x, ...}

C_3 \$ f_3 = {1xxxxx, ...}

.
.
.

C_6 # f_6 = {xxx0x1, xxx1x0, ...}.

Thus, two fault-free 4-subcubes exist that can be used to reconfigure B_6. In this example, most of the details are suppressed to maintain the readability of the paper. We shall give the details shortly.

Theorem. For any n-dimensional hypercube in which we are given a list of faulty subcubes and/or links, the non-faulty subcube can be identified using the algorithm.

Proof: Refer to the appendix. ∎

The algorithm described here is good for computer simulation. An algebraic technique formulated using these concepts is presented next. To help understand the technique we need the following representations and definitions.

P1. We represent a fault f_i using notation {a, $\bar{a}$, a} where an uncomplemented (complemented) variable denotes 1 (0). An a denotes the "q" of link failure. An absent variable in a position represents the "x". Therefore, a faulty node 0110 is represented by $\bar{a}$ b c $\bar{d}$, a subcube fault 11x by ab, and a link fault 001q10 by $\bar{a}$ $\bar{b}$ c a e f.

P2. A D-operator operates on f_i. It modifies f_i using following transformations:

AND(·) → OR(+)
a → $\bar{a}$
$\bar{a}$ → a
a → (a + $\bar{a}$)

Thus, D($\bar{a}$ b c $\bar{d}$) = (a + $\bar{b}$ + $\bar{c}$ + d) and D($\bar{a}$ $\bar{b}$ c a e $\bar{f}$) = (a + b + $\bar{c}$ + d + $\bar{d}$ + $\bar{e}$ + f). Note, we have used · (+) and ∩ (∪) interchangeably. Also the Boolean identity of the type a + $\bar{a}$ ≠ 1 in this case. However, the identity a $\bar{a}$ is still applicable.

P3. The "o" operator ("o" stands for # or \$) is, then, described as:

$C_2 = D(f_1)$ and (4)
$C_{i+1} = C_i\, D(f_i)$.

Here, juxtaposition of C_i with $D(f_i)$ denotes the Boolean AND operation and $C_i\, D(f_i)$ can be expanded using Boolean rules [15]. Equation (4) has the same effect as Equation (3).

Using P1 through P3 above, the illustrating example is solved as follows.

f_i	Binary representation	Variable representation
f_1	0 1 1 0 0 0	$\bar{b}_5$ b_4 b_3 $\bar{b}_2$ $\bar{b}_1$ $\bar{b}_0$
f_2	0 0 0 1 0 1	$\bar{b}_5$ b_4 b_3 b_2 b_1 b_0
f_3	0 0 1 q 1 0	$\bar{b}_5$ $\bar{b}_4$ b_3 b_2 b_1 $\bar{b}_0$
f_4	1 0 0 0 1 0	b_5 $\bar{b}_4$ $\bar{b}_3$ b_2 b_1 $\bar{b}_0$
f_5	1 1 0 1 1 1	b_5 b_4 $\bar{b}_3$ b_2 b_1 b_0
f_6	1 0 1 1 0 1	b_5 $\bar{b}_4$ b_3 b_2 $\bar{b}_1$ b_0

(*i*) $C_2 = D(f_1) = (b_5 + \bar{b}_4 + \bar{b}_3 + b_2 + b_1 + b_0)$

(*ii*) $C_3 = C_2 D(f_2) = (b_5 + \bar{b}_4 + \bar{b}_3 + b_2 + b_1 + b_0)$

$(b_5 + b_4 + b_3 + \bar{b}_2 + b_1 + \bar{b}_0)$

$= b_5 + b_1 + \bar{b}_4 b_3 + \bar{b}_4 \bar{b}_2 + \bar{b}_4 \bar{b}_0 + b_4 \bar{b}_3 + \bar{b}_3 \bar{b}_2$

$+ \bar{b}_3 \bar{b}_0 + b_4 b_2 + b_3 b_2 + b_2 \bar{b}_0 + b_4 b_0 + b_3 b_0 + \bar{b}_2 b_0$

(iii) $C_4 = C_3 D(f_3)$

$= c_3(b_5 + b_4 + \bar{b}_3 + b_2 + \bar{b}_2 + \bar{b}_1 + b_0)$

$= b_5 + b_4\bar{b}_3 + b_4b_2 + \bar{b}_4\bar{b}_2 + b_4b_1 + b_4b_0$

$+ \bar{b}_4\bar{b}_0 + \bar{b}_3\bar{b}_2 + b_3b_2 + \bar{b}_3b_1 + \bar{b}_3\bar{b}_0 + b_3b_0$

$+ b_2b_1 + \bar{b}_2b_1 + b_2\bar{b}_0 + \bar{b}_2b_0 + b_1b_0$

$+$ *higher order terms*

(iv) $C_5 = C_4 D(f_4) = b_5b_4 + b_5b_3 + b_5b_2 + b_5\bar{b}_1$

$+ b_5b_0 + b_4\bar{b}_3 + b_4b_2 + b_4b_1 + b_4b_0 + b_3b_2$

$+ b_3b_0 + b_2b_1 + b_2\bar{b}_0 + \bar{b}_2b_0$

$+$ *higher order terms*

(v) $C_6 = C_5 D(f_5) = b_5b_3 + b_3b_2 + b_3b_0 + \bar{b}_2b_0$

$+ b_5\bar{b}_1 + b_2\bar{b}_0 +$ *higher order terms*

(vi) $C_7 = C_6 D(f_6)$

$= \bar{b}_2b_0 + b_2\bar{b}_0 +$ *higher order terms*

The cubes $\bar{b}_2\, b_0$ and $b_2\, \bar{b}_0$ in binary representation give the results as (xxx0x1, xxx1x0). While computing the fault-free 4-subcubes we have suppressed the information regarding higher order terms. These are relevant only when we fail to compute a non-faulty 4-subcube. The notion may be extended to enumerate fault-free d-subcubes. Here, we should initially use intermediate terms having cardinality j, $1 \le j \le d$. Notes that terms with cardinality j correspond to n-tuples with j bound coordinates and n−j free coordinates. Terms with cardinality (d+1) or higher are kept with a different group/bin. This bin is, obviously, useful when we are unable to generate a d-subcube. In this way, we are using a pruned-tree approach to contain the size of intermediate terms generated by the algorithm.

5. Complexity analysis

We now analyse the time complexity of the algorithm presented in Section 4. We assume that computing $c_T \# f_S$ for one cube c_T and for one cube f_S can be done in one time step for each resulting cube. We assume that each fault is a 0-subcube (that is, a node) as this will produce the worst case bounds. For a given list of m faults, the algorithm computes $C_{i+1} = C_i \circ f_i$ on each of m iterations. Each C_i may be a set of cubes, so our object is to bound the number of cubes in C_i. Initially, $C_1 = \{x\ x\ x\ \ldots\ x\}$. By definition of the # and S operators, in the worst case, C_2 may contain n cubes: $x\ x\ \ldots\ x\ \alpha_0$, $x\ x\ \ldots\ x\ \alpha_1\ x$, ..., $\alpha_n\ x\ \ldots\ x$, where $\alpha_i \in \{0, 1\}$. In the worst case, a set of cubes produced by $c_T \# f_S$ may contain at most as many cubes as there are x's in c_T, and each of those cubes will contain one less x than c_T. Hence, C_i may contain at most $n(n-1)\cdots(n-i+1) = n!/(n-i)!$ cubes, where n is the dimension of the hypercube B_n under consideration. Actually, with n symbol positions and 3 possible symbols, {0, 1, x}, there are at most 3^n possible cubes. Let v be the least value of i such that $n!/(n-i)! \ge 3^n$. Note that $n(n-1)\cdots(n-i+1) < n^i$. So the time to compute m iterations of the algorithm, for $m \le v$, is

$$\sum_{i=1}^{m} \frac{n!}{(n-i)!} < \sum_{i=1}^{m} n^i = O(n^m).$$

And the time to compute m iterations of the algorithm, for $m \ge v$, is

$$\sum_{i=1}^{v} \frac{n!}{(n-i)!} + \sum_{i=v+1}^{m} 3^n$$

$$\le \sum_{i=1}^{v} n^i + \sum_{i=v+1}^{m} 2^{O(n)}$$

$$= O(n^v) + (m-v)2^{O(n)}$$

$$< m2^{O(n)}.$$

In terms of N, the size of the hypercube, the time complexity for number of faults $m \le v$ is $O((\log N)^m) = O(N)$, and the time complexity for $m > v$ is $O(mN)$.

The above time complexity describes the time to generate a list of all fault-free subcubes, given m failed nodes. If we wish instead to compute a list of all fault-free subcubes of dimension at least n−k, then we can obtain a better time complexity. In this instance, the cubes with k or fewer 0 or 1 symbols in their representations (that is, k or fewer bound coordinates) correspond to cubes of dimension n−k or higher. If $m \le k$, then we again obtain a time complexity of $O(n^m)$. But if $m > k$, then we obtain the following time complexity.

$$\sum_{i=1}^{k} \frac{n!}{(n-i)!} + \sum_{i=k+1}^{m} \frac{n!}{(n-k)!}$$

$$< \sum_{i=1}^{k} n^i + \sum_{i=k+1}^{m} n^k$$

$$= O(n^k) + O((m-k)n^k)$$

$$= O((m-k)n^k).$$

This time improves on Özgüner and Aykanat's algorithm [4] that requires $O\left(mk\binom{n}{k}\right)$ time to locate the available subcubes of dimension n–k or greater.

The ATARIC procedure will have a significantly better expected time complexity, however, for a random set of faults. Let us call a cube with k bound coordinates and n–k free coordinates as a type k cube. Each cube in set C_i can be of type j, for $1 \le j \le i-1$. The analysis above assumed that each type k cube would in a single iteration fragment into k type k–1 cubes. By the definition of the # and $ operators, a type k cube will either fragment into k type k–1 cubes or remain as a single type k cube or disappear. A type k cube will only fragment if each of the k 0's or 1's in the c_T cube matches exactly with k identical 0 or 1 bits in the faulty element f_S; otherwise, the c_T cube remains as a single cube or disappears. For a random fault f_S, the probability is 2^{-k} that the cube c_T will fragment and $1-2^{-k}$ that it will remain as the same single cube or disappear. Therefore, the number of cubes in C_m is much, much less than $O(n^m)$.

6. Conclusion

This paper has described two operators, namely # and $. The sharp (#) operator is used extensively for generating test set for logical faults in PLA and also for reliability evaluation in general networks. The dollar ($) operator is introduced in this paper for the first time. These two operators are the main features of ATARIC. The proposed technique is straightforward and efficient as compared to previous algorithms [4,8]. We plan to extend the concept for the subcube allocation and task migration problems in hypercube multiprocessors.

References

[1] S. Rai and D. P. Agrawal, *Advances in distributed system reliability*, IEEE Computer Society Press, Washington D.C., 1990 (tutorial text).

[2] S. B. Akers and B. Krishnamurthy, "A group theoretic model for symmetric inter- connection network," *IEEE Trans. Computers*, vol. C-38, Apr. 1989, pp. 555-566.

[3] J. P. Hayes and T. Mudge, "Hypercube supercomputers," *IEEE Proceedings*, vol. 77, Dec. 1989, pp. 1829-1841.

[4] F. Özgüner and C. Aykanat, "A reconfiguration algorithm for fault tolerance in a hypercube multiprocessor," *Information Processing Letters*, vol. 29, 1988, pp. 247-254.

[5] J. R. Armstrong and F. G. Gray, "Fault diagnosis in n-cube array of microprocessors," *IEEE Trans. Computers*, vol. C-30, Aug. 1981, pp. 587-590.

[6] K. V. Bhatt, "An efficient approach for fault diagnosis in a Boolean n-cube array of microprocessors," *IEEE Trans. Computers*, vol. C-32, 1983, pp. 1070-1071.

[7] W. J. Dally, "Performance analysis of k-ary n-cube interconnection network," *IEEE Trans. Computers*, vol. C-39, Jun. 1990, pp. 775-785.

[8]. B. Becker and H. U. Simon, "How robust is the n-cube?," *Proc. 27th IEEE Symposium on Foundations of Computer Science*, Oct. 1987, pp. 283-291.

[9] S. Abraham and K. Padmanabhan, "Reliability of the hypercube multicomputers" *Proc. 1988 International Conference on Parallel Processing*, Aug. 1988, pp. 90-93.

[10] W. Najjar and J.-L. Gaudiot, "Network resilience. A measure of network fault tolerance," *IEEE Trans. Computers*, vol. C-39, Feb. 1990, pp. 174-181.

[11] J. Kim *et al.*, "Reliability evaluation of hypercube multicomputers," *IEEE Trans. Reliability*, vol. R-38, Apr. 1989, pp. 121-129.

[12] H. K. Regbati, "Fault detection in PLAs," *IEEE Design and Test*, Dec. 1986, pp. 43-50.

[13] M. Veeraraghavan and K. Trivedi, "An improved algorithm for the symbolic reliability analysis of networks," *Proc. 9th Symposium on Reliable Distributed Systems*, Oct . 1990, pp. 34-43.

[14] S. Soh and S. Rai, "CAREL: Computer aided reliability evaluator for distributed computer networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 2, Apr. 1991, pp. 199-213.

[15] R. Miller, *Switching theory, volume I: Combinational circuits*, Wiley, New York, 1965.

[16] F. Harary, "A survey of the theory of hypercube graphs," *Compu. Math. Applications*, vol. 15, 1988, pp. 277-289.

[17] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Computers*, vol. C-37, July 1988, pp. 867-871.

[18] A.-H. Esfahanian, "Generalized measures of fault tolerance with application to n-cube networks," *IEEE Trans. Computers*, vol. C-38, Nov. 1989, pp. 1586-1591.

[19] A. Grnarov, L. Kleinrock, and M. Gerla, "A new algorithm for network reliability computation," *Computer Networking Symposium*, Dec. 1979, pp. 17-20.

Appendix

Proof of correctness

The following three steps are useful to verify the correctness of ATARIC:

Step 1. If we form $c_i \# f_i$, where c_i and f_i are cubes, then the sharp (#) operator produces the set of subcubes of c_i which do not intersect with f_i. If a y is obtained in any coordinate position, then that coordinate is bound to 0 (1) in c_i and 1 (0) in f_i, so the cubes c_i and f_i are disjoint, and the sharp operator produces c_i.

Step 2. The removal of a link in dimension i from a cube produces the following subcubes: the set of subcubes produced on the removal of the 1-subcube containing the link (that is, the link and its adjacent nodes) and the two subcubes produced by partitioning the original cube along dimension i. These last two subcubes have the same description as the original cube, except for a 0 or 1 in position i. If we form $c_i \$ f_i$, where c_i is a cube and f_i is a link, then the $ operation is very similar to the # operation. The difference lies in the handling of the variable t that may occur in the same coordinate position as the q in f_i. The case in which t is set to z produces the subcubes disjoint from the 1-subcube containing link f_i; the cases in which t is set to 0 and 1 produce the subcubes that contain the endpoints of the link.

Step 3. The result of the operation in Equation (3) is a set of cubes from C_1 that are not covered by f_i. The iterative application gives a cover of the subcubes, if any, not covered by the fault list.

Table 1. Coordinate # and $ operations

a_i \ b_i	0	1	x
0	z	y	z
1	y	z	z
x	1	0	z

(a) # operation

a_i \ b_i	0	1	q
0	z	y	y
1	y	z	y
x	1	0	t

(b) $ operation

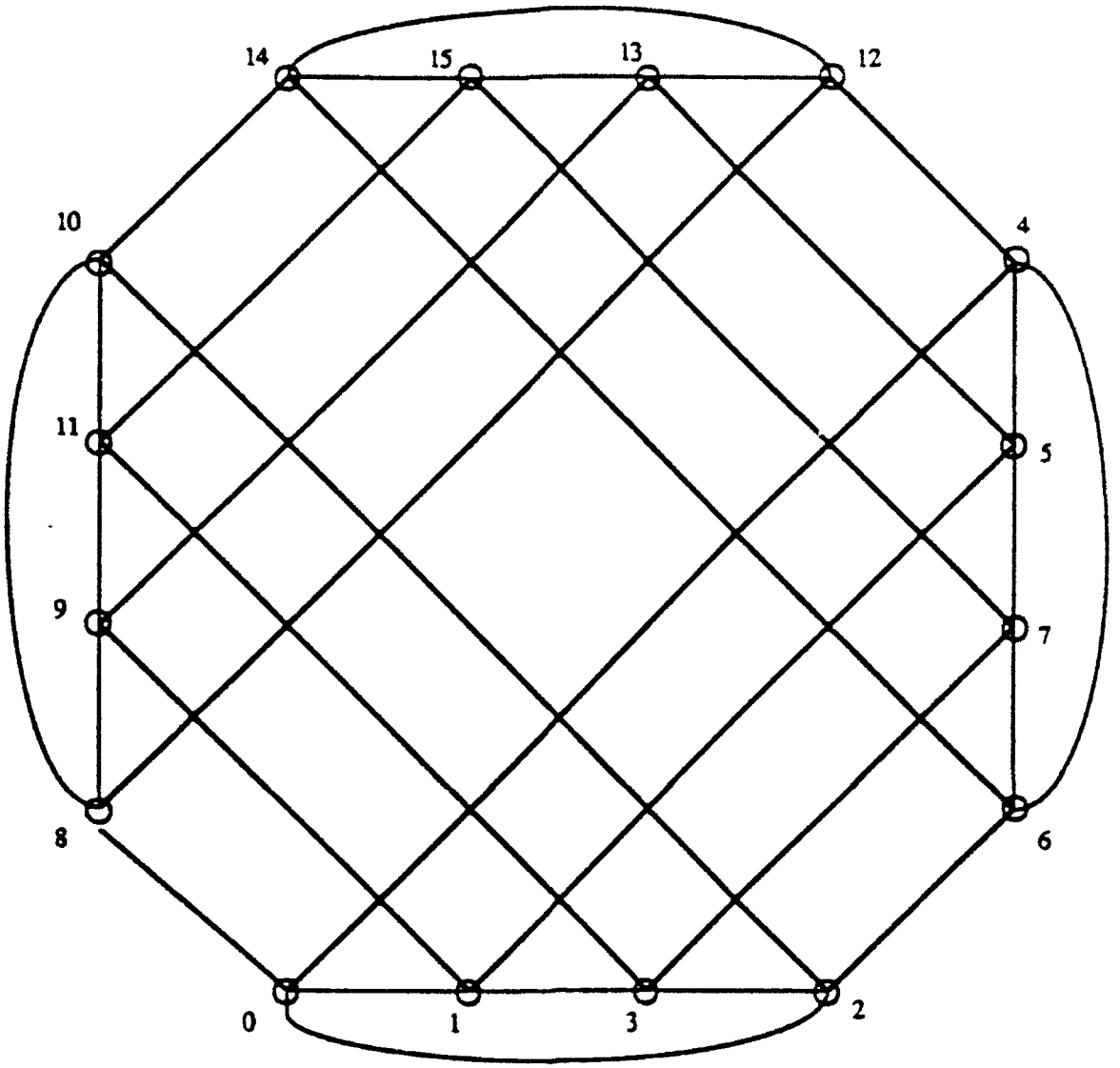


Figure 1. A 4-hypercube

IEEE TRANSACTIONS ON

PARALLEL AND DISTRIBUTED SYSTEMS

APRIL 1991 VOLUME 2 NUMBER 2 (ISSN 1045-9219)

A PUBLICATION OF THE IEEE COMPUTER SOCIETY

CAREL: Computer Aided Reliability Evaluator for Distributed Computing Networks

Sieteng Soh, *Student Member, IEEE*, Suresh Rai, *Senior Member, IEEE*

Abstract— This paper presents an efficient method to compute the terminal reliability (the probability of communication between a pair of nodes) of a distributed computing system (DCS). We assume that the graph model $G(V, E)$ for DCS is given. Also, it is assumed that we have path and/or cut information for the network $G(V, E)$. Boolean algebraic concepts are used to define four operators namely, COMpare, REDuce, CoMBine, and GENerate. The proposed method, henceforth called CAREL (Computer Aided RELiability evaluator), uses these four operators to generate exclusive and mutually disjoint ($e.m.d.$) events, and hence the terminal reliability expression. Examples illustrating the technique are given in the text. We have implemented CAREL using bit vector representation [11] on Encore MULTIMAX 320 system. CAREL solves large DCS networks (having pathset of the order of 780 and cutset of the order of 7300 or more) with reasonable memory requirement. A comparison with existing algorithms reveals the computational efficiency of the proposed method. The proof of correctness of CAREL is included in the Appendix.

Index Terms— Bit vector representation, Boolean technique, CAREL, combinational and sequential reliability, distributed system reliability, minimal conditional cube, minpath, mincut, operators—COM, RED, CMB, and GEN, reliability evaluation tool.

I. Introduction

ADVANCES in computer technology and the need to have the computers communicating with each other have led to an increased demand for a reliable distributed computing system (DCS). An important performance metric in the design of highly reliable DCS network is provided by its terminal reliability parameter [1]. Note, the terminal reliability refers to the probability that at least one path exists between a prescribed node pair in the distributed system [2], [6]. All methods of terminal reliability computation are known to be NP-hard [2], [9], [10], [23], [26], [27].

Several algorithms dealing with the terminal reliability evaluation are proposed in the literature [1]–[4], [7], [12]–[19], [21], [24], [25]. These methods fall in any one of the following categories: state enumeration, decomposition technique, inclusion–exclusion, factoring, and sum of disjoint products. A summary of these techniques, including their relative merits and demerits, can be found in [2].

Various techniques [1], [3], [12]–[16], [18], [19] have utilized Boolean concepts to obtain a sum of disjoint products, and hence the terminal reliability parameter of a given DCS network. All of these methods start with a Boolean polynomial formed by either the success terms (minimal paths) or the failure terms (minimal cuts) for a given DCS. The paths or cuts are sequenced in order of their increasing cardinality. For each group of terms of the same size, the ordering is lexicographic following the orders of the symbols of the alphabets. The ordering of terms helps reduce the overall time complexity for generating sum of disjoint products (SDP) expression. A method [1], [3], [12]–[16], [18], [19], then converts the Boolean polynomial of paths or cuts into an equivalent Boolean SDP form that represents the disjoint system logic. Note, an SDP expression has 1:1 correspondence with the system probability formula. A drawback of the algorithms based on the manipulation of Boolean sum of products or implicants is in the iterative application of certain operations and the fact that the Boolean function changes at every step and may be clumsy. Moreover, the Boolean function is simplified using absorption rules [20] and, thus requires a considerable computational effort [3]. Therefore, most sum of disjoint products algorithms are applicable only to small to moderate sized networks. Recently, Veeraraghavan and Trivedi [18] has reported an algorithm modifying the concepts given in [3]. Their method solves a large DCS network (Fig. 11 in [18] is same as our Fig. 19 but for few directional links and, thus, has only 425 paths) in 166 668 s (= 46.3 CPU hours). Obviously, we still need an efficient algorithm (on the application of Boolean algebra) to solve terminal reliability problem in large distributed processing system. The CAREL (Computer Aided RELiability evaluator) provides a solution in this direction. CAREL computes the terminal reliability parameter of the DCS network of Fig. 19 in less than a minute CPU time.

The CAREL uses Boolean algebraic concepts, and COMpare, REDuce, CoMBine, and GENerate operators. Refer to the text for a discussion on these operators. The algorithm is efficiently implemented as the CPU time obtained for generating terminal reliability parameter for some moderate to large sized networks is considerably less as compared to that reported for other algorithms [1], [3], [18]. SYREL [1] provides an efficient implementation scheme for E-operator technique[12] using set theoretic concepts, minimal conditional sets (MCS), and partitioning MCS's into independent and dependent groups to reduce the amount of computation in generating disjoint sum of products expression. The # (sharp) and @ operators [3], [18] reduce the total number of disjoint products by grouping variables together such that approximately 30–40 % saving in the final reliability expression is achieved. CAREL utilizes the advantages of both SYREL [1] and [3] and [18] to obtain low computation time. Moreover, CAREL operators are bit implementable (refer to text).

The layout of the paper is as follows. Section II provides a generalized view of various existing Boolean techniques. It outlines and compares their basic philosophies. The notion of data representation is also considered. Section III introduces the notation used and defines four operators namely COM, RED, CMB, and GEN. The algorithm, its implementation details, and various illustrating examples, are described in Section IV. Section V provides comparison tables showing the computer time for evaluating moderate to large sized DCS networks. It also presents a comparison of CAREL with existing techniques. We conclude the paper in Section VI. The Appendix shows the proof for the

Manuscript received October 6, 1989; revised July 24, 1990.

The authors are with the Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803.

IEEE Log Number 9042573.

1045-9219/91/0400-0199$01.00 © 1991 IEEE

correctness of the algorithm.

II. Preliminaries

Consider a linear graph $G(V, E)$ model for the DCS network such that nodes V (edges E) represent computers (communication links). Assume $G(V, E)$ is free from self loops and directed cycles. Each edge has two states : good (UP) or bad (DOWN). Nodes are perfect (imperfect nodes can be considered following a method given in [2]). Let the failures be statistically independent (statistically dependent failures can be solved following a method given in [5]). This assumption is useful to make the problem mathematically tractable. A minpath P_i is a path from a source node s to a terminal node t in $G(V, E)$. It is formed by the set of UP edges such that no nodes are traversed more than once. Pathset is defined as the set of minpaths. A cut is a disconnecting set. All communication between a prescribed (s, t) node pair is disrupted once the edges in (s, t) cut fail. Define a *mincut* as a cut which has no proper subset that is also a cut, and cutset as the set of mincuts. Assume that either pathset or cutset between a source s and terminal t in $G(V, E)$ is known.

A. Data Representation

Data structure is an important aspect of designing efficient algorithms [8]. Rosenthal [11] discussed the advantage and disadvantage of three different kinds of data representation for cutset enumeration. This section briefly describes one of the representations, namely the bit vector representation because it is suitable for the implementation of the proposed method. A minpath (mincut) in a network with l links is represented by l bits. An UP link of the network is denoted by a binary 1. A binary 0 stands for a *don't care* state (not a DOWN state). Consider the minpaths *ab*, *cd*, *ade* and *bce* between the (s, t) node pair in Fig. 1. These minpaths are stored in memory as

ab : 0000000000000011 *cd* : 0000000000001100
ade : 0000000000011001 *bce* : 0000000000010110.

In this example, we have utilized the word size w as 16 bits. Note, a minpath (mincut) requires $\lceil l/w \rceil$ words of memory. With bit vector representation, the storage requirement for a minpath (mincut) depends on the total number of links in the network and not on the size of the pathset (cutset). Coding and decoding of path information into bit representation and vice versa may add extra cost, as it involves l bit testings. However, this pre- and postprocessing of minpaths (mincuts) are one time operation. They are usually worth the extra computation as the generation of disjoint events requires considerable manipulations. Moreover, the ability of bit representation in detecting and eliminating redundant terms using set theoretic operations like union, intersection, subset, etc., is an important advantage. To illustrate the concept for redundancy checking, assume the reference term X, and a test term XY (which is a redundant subset of X):

```
reference (X)   1 1 0 0 1
test      (XY)  1 1 1 0 1   ;OR operation
                1 1 1 0 1
test      (XY)  1 1 1 0 1   ;EOR operation
                0 0 0 0 0
```

A result "0 0 0 0 0" shows XY redundant. A duplicate term is detected using the same approach. The set theoretic operations are implemented easily and the computation time is independent of the size of the network. The number of the links l, which affects the speed of the set operations, increases the computation time by one unit for every w additional links. The proposed method uses these operations with COMpare and REDuce operators (refer to the text) and we have discussed it in detail in Section IV-B while considering the implementation of various set theoretic operations.

B. Boolean Techniques Concept

Various Boolean techniques of reliability evaluation start with a sum of products expression for minpaths or cutsets and convert it into an equivalent sum of disjoint products expression. In the SDP form, an UP or logical success (DOWN or failure) state of a link x is replaced by link reliability p (unreliability q), and the Boolean sum (product) by the arithmetic sum (product). In other words, the SDP expression is interpreted directly as an equivalent probability expression of terminal reliability. If F_i represents a path identifier (an UP state of a link in a path P_i has 1 in F_i, while a *don't care* is represented by 0), the sum of products expression F is given by

$$F = \bigcup_{i=1}^{n} F_i \tag{1}$$

where n denotes the number of minpaths (cutsets) between (s, t) node pair in $G(V, E)$. Equation (1) is modified either canonically or conservatively to generate the equivalent SDP expression, F(disjoint). The conservative modification is usually preferred, since it is more efficient compared with canonical modification, where 2^l events are required to determine F(disjoint). (l is the number of links in the network.) A simple way to generate the mutually disjoint events in (1) is as follows:

$F_1 + F_2\overline{F_1} + F_3\overline{F_1}\,\overline{F_2} + \cdots + F_n\overline{F_1}\,\overline{F_2}\cdots\overline{F_{n-1}}$ where $\overline{F_i}$ denotes DOWN events of path P_i. The probability of UP (operational) for an ith term $F_i\,\overline{F_1}\,\overline{F_2}\cdots\overline{F_{i-1}}$ can be evaluated using conditional probability and standard Boolean operations as

$$\Pr(F_i)\cdot\Pr\left(\overline{F_1}\cdot\overline{F_2}\cdots\overline{F_{i-1}}\,|F_i\right) = \Pr(F_i)\prod_{j=1}^{i-1}\Pr(E_j).$$

Here, an E_j represents a conditional cube [20] and defines conditions for a path identifier F_j DOWN given F_i as UP (operational). The probability of the first event $\Pr(F_1)$ can be determined in a straightforward manner, since the failures are assumed to be statistically independent. However, the coefficient $\Pr(E_j)$ requires further consideration since various terms within E_j's will, in general, be not disjoint [2]. This necessitates E_j's to be made mutually disjoint before we generate the equivalent probability expression. Note, F_i's in (1) are sequenced in the order of their increasing cardinality, and also for each group of terms of the same size, the ordering is lexicographic. Therefore, the disjoint products for any $(m-1)$ size path identifier F_j, where m denotes the number of nodes in $G(V, E)$, is obtained directly by intersecting the complements of the remaining $l-(m-1)$ links of $G(V, E)$ with F_j. This observation (first made in [12], and then proved in [1]) reduces computational time for algorithms based on Boolean concepts.

Various researchers [2] have given techniques which generate a disjoint expression for (F_i, F_k) pairs in (1), and also the (E_j, E_l) terms within an F_i. The following three propositions (P I through P III) that convert F into F(disjoint) represent basic philosophies for most Boolean methods in the reliability literature

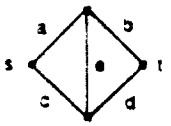


Fig. 1. Bridge network.

P I: The proposition P I defines intermediate term(s) T_i's as

$$T_i = \bigcup_{j=1}^{i-1} F^j | \text{each literal of } F_{i-1} \tag{2}$$

where $F^1 = F_1$ and $F^i = F_i\ OP_1\ T_i$. Here, F^i refers to the equivalent disjoint product term(s) for F_i. The operation "OP_1" is a necessary disjointing operator. (Table I lists various operators.) The F(disjoint) expression is, then, given by

$$F(\text{disjoint}) = \bigcup_j F^j. \tag{3}$$

Algorithms [16] and [19] make use of proposition P I.

P II: For each term $F_i, 1 < i \leq n, T_i$ is defined to be the union of all predecessor terms $F_1, F_2, \cdots, F_{i-1}$, in which any literal that is present in both F_i and any of the predecessor terms is deleted from those predecessor terms, i.e.,

$$T_i = \bigcup_{j=1}^{i-1} F_j | \text{each literal of } F_{i-1}. \tag{4}$$

Consider $F^1 = F_1$, and define $F^i = F_i\ OP_2\ T_i$. Equation (3), then, obtains the equivalent F(disjoint) expression. Hariri and Raghavendra [1], Rai and Aggarwal [12], Fratta and Montanari [14], and Bennetts [10] have based their methods on proposition P II. Refer to the Table I for OP_2 operator.

P III: For $1 < j \leq n$, use operation "OP_3" to perform

$$F^j = (\ldots((F_j\ OP_3\ F_1)OP_3\ F_2)\ OP_3 \ldots)\ OP_3\ F_{j-1}. \tag{5}$$

Equation (5) obtains a set of disjoint cubes corresponding to F_j. Note, $F^1 = F_1$, and OP_3 represents an appropriate disjointing operator. The F(disjoint) expression is, then, given by (3). Tiwari and Verma [22], Grnarov *et al.* [3], Abraham [13], and Veeraraghavan and Trivedi [18] have proposed their techniques using P III concept. References [3], [13], [15], and [22] consider F_j's in cubical notation [20]. Table I lists operation OP_3 as suggested by various researchers in the literature [3], [13], [15], [18], [22].

Example: To illustrate propositions P I through P III, consider a DCS network shown in Fig. 1. For the (s, t) node pair, there exists four minpaths: ab, cd, ade, bce. In what follows, we explain steps to generate the exclusive and mutually disjoint event(s) or cube(s) for F_3 (= ade). This is demonstrated using typical methods for propositions P I through P III. For uniformity, we keep the notation [12] with a modification that the state of a DOWN link $\bar{y}$ is denoted as $(1 - y)$.

P I: Considering [16], $F^1 = ab$, and $F^2 = (1 - ab)cd$. Use (2) to define T_3 as $(b + (1 - b)c)$ for F_3. Replacing OP_1 with X-operator [16], $\mathbf{X}\ (T_3)$ is $(1 - b)(1 - c)$. F^3 is then obtained as $F^3 = F_3\ \mathbf{X}\ (T_3)$. The term F^4 is generated similarly. An expression for F(disjoint)is $F(\text{disjoint}) = ab + (1 - ab)cd + (1 - b)(1 - c)ade + (1 - a)(1 - d)bce$.

TABLE I
OP_1–OP_3 USED WITH BOOLEAN ALGEBRAIC TECHNIQUES
(↔ Proposed algorithm)

Operator	Function	Reference
OP_2	Cutset Disjoint Procedure	[1]
OP_3	Modified $ operator	[18]
OP_3	$ operator	[3]
OP_2	E-operator	[12]
OP_3	COMPARE( ) function	[13]
OP_1	X-operator	[16]
OP_2	Boolean negation	[22]
OP_2	Relative complement and Procedure 1	[10]
OP_1, OP_2	CMB (*) operator	↔

P II: We have used E-operator [12] to explain the operation OP_2 and, hence, the concept behind proposition P II. The terms $F^1 = ab$, and $F^2 = ((1 - a) + a(1 - b))cd$ are computed using [12]. To generate F^3, an intermediate term T_3 is obtained as $T_3 = ab + cd\,|_{a=d=e=1} = b + c$

Note, $\mathbf{E}\ (T_3)$ is $(1 - b)(1 - c)$. Hence, $F^3 = F_3\ \mathbf{E}\ (T_3)$. Similarly, we obtain F^4.

Equation (3) is

$F(\text{disjoint}) = ab + ((1 - a) + a(1 - b))cd + (1 - b)(1 - c)ade + (1 - a)(1 - d)bce$

P III: Use [3] to obtain the terms $F^1 = ab$, and $F^2 = (1 - ab)cd$. Here, the $ (sharp) operator [3] substitutes for OP_3. The cube F^3 is generated using $F^3 = ((F_3\ OP_3\ F_1)\ OP_3\ F_2)$. The inner term $F_3\ OP_3\ F_1$ gives $(1 - b)\ ade$, which with F_2 generates $(1 - b)(1 - c)ade$. Similarly, compute F^4 for F_4. Equation (3), then, gives

$F(\text{disjoint}) = ab + (1 - ab)cd + (1 - b)(1 - c)ade + (1 - a)(1 - d)bce$.

Note, F(disjoint) expression obtained from different propositions when expanded out should be identical. In Fig. 1, the terminal reliability is 0.97848 when each link is assumed to have a reliability of 0.9.

C. Existing Boolean Techniques—A Comparison

Propositions P I through P III maintain the minpaths or mincuts list in memory (1). Consider 1 for UP link and 0 for *don't care*, and utilize bit representation technique (discussed in Section II-A). The memory requirement is, then, $\lceil l/16 \rceil$ words per path (cut), where l is the number of links in the DCS network $G(V, E)$. Proposition P I makes F_i disjoint with respect to $\cup_{j=1}^{i-1} F^j$, while propositions P II and P III utilize $\cup_{j=1}^{i-1} F_j$. Should we have similar operations to implement (2) and (4), the proposition P I will require more operations than that needed for P II. Generally, an F_i generates more than one *e.m.d.* events F^i's. Hence, the number of events involved in $\cup$, F^j is larger than that in $\cup$, F_j. For example, Table IV ($N_{80}^{12,6}$) shows results for $i = 780$. The number of terms in (2) is more than 50 000; on the other hand, (4) needs exactly $(i - 1)$, i.e., 779 terms. Note, in proposition P I, the generated *e.m.d.* events have to be kept in the memory to implement (2), which is not the case for P II or P III. This

makes proposition P I sequential. Moreover, P I demands a huge memory space to evaluate a large DCS networks. On the other hand, P II or P III has implicit parallelism, making it easier for the programmers to implement them on parallel systems. Overall, the proposition P II or P III provides advantages in comparison with P I.

An analysis of performance comparison between a typical example of proposition P II and P III is discussed in [1]. SYREL [1], an implementation technique for E-operator [12], is shown to have better performance in comparison with S -operator [3]. It means proposition P II outperforms P III. Moreover, proposition P II offers a faster implementation approach than that in P I or P III. The bit vector implementation of F_i makes the realization of (4) w (word size) times faster than generating (2) based on proposition P I. Later in Section V, we shall show that CAREL outperforms E-operator [12] or SYREL [1].

III. CAREL (Computer Aided Reliability Evaluator): Background

Section III-A presents a notational concept which is useful to describe CAREL. We have defined COMpare, REDuce, CoMBine, and GENerate operators in Section III-B. These four operators form a basis for our algorithm CAREL.

A. Notation

For the DCS network $G(V, E)$, consider a set of paths P_i's between source s and destination t. The path identifier F_i identifies P_i in the cubical notation [20] using a string of symbols {0, 1}. Thus, an UP state of a link in the P_i has 1 in F_i, while a *don't care* state is represented by a 0. F^j denotes exclusive and mutually disjoint event(s) and is generated for F_i. To help obtain F^j, conditional cubes E_i's (defined later) are utilized. Both E_i and F^j are in Boolean domain. An E_i is composed of complemented and absent variables and requires $\{-\beta^l, 0\}$ symbols, while F^j uses $\{-\beta^l, 0, 1\}$ [19]. The β represents a positive integer and l is a superscript or index. We use a superscripted negative integer to represent a complemented variable (DOWN state of a link). To help illustrate the concept of this new notation, the cube $\overline{ab}\,c\,\overline{d}$ is represented as (-2^1 -2^1 1 -1^2). Similarly, the product term $\overline{abd}\;\overline{ce}\;g$ is denoted as ($-3^1 - 3^1 - 2^2 - 3^1 - 2^2$ 0 1). Note the following:

1) An uncomplemented, or complemented, or absent variable is replaced by 1, $-\beta^l$, or 0 in the position of the variable, respectively.
2) A "$-\beta^l$" represents complements of β^l number of variables which are grouped together. (-1 signifies single variable complement. Use of index is optional with -1.)

The need of superscript or index is illustrated with the example of a Boolean term $\overline{w_1w_2}\;\overline{w_3w_4}\;\overline{w_5w_6}$ represented as ($-2^1 - 2^1 - 2^2 - 2^2 - 2^2 - 2^2$). If the indexes are not used, a notation of (-2 -2 -2 -2 -2 -2) for this example, in all likelihood, would be wrongly interpreted. The advantage of ($-\beta^l$, 0, 1) notation lies in its uniqueness in handling complemented variables that are grouped together [3], [18], [19].

B. CAREL Operators

In what follows, we discuss four operators COM, RED, CMB, and GEN used in CAREL. Given a set of path identifiers $\cup_{i=1}^{n} F_i$, we want to generate disjoint events F^j generated from the F_i for all i. The COM operator generates a set of conditional cubes E_{ij}, $j = 1, \cdots, i-1$ for an F_i. The conditional cube set E_i's describes the events that a path identified by F_i is operational, while paths P_j, $j = 1, 2, \cdots, i-1$ fail. The RED operator, on the other hand, is used to remove the redundant conditional cubes from the generated set E_i's. We call the nonredundant cubes minimal conditional cubes (MCC). If we have only one cube in the MCC set, or the cubes are mutually disjoint among themselves, we may generate the disjoint events F^j directly. But, in general, the MCC's are not disjoint. Thus, we need the CMB operator to create disjoint events of MCC's. Refer to Section II-B for the need of making MCC's E_i's mutually disjoint. Finally, the last operator, GEN, gives the disjoint events F^j.

1) COM (\ ) operator: The COM operator is used to compare two cubes. The COM1 (COM2) version describes proposition P I (P II). For P I, consider two cubes $F_k = (a_1\, a_2 \cdots a_l)$ and $F^j = (b_1\, b_2 \cdots b_l)$ where $a_i \in \{0, 1\}$ and $b_i \in \{-\beta, 0, 1\}$. The COM1 (\) is, then, defined as

$$F_k \backslash F^j = \begin{cases} \phi: & \text{if } a_i = 1 \text{ in all those } \beta^l \text{ positions where } b_i\text{'s} \\ & \text{are } -\beta^l. \text{ It shows that } F_k \text{ and } F^j \text{ are} \\ & \text{mutually disjoint.} \\ E_k: & \text{otherwise} \end{cases}$$

where, the conditional set E_k is $(c_1\, c_2 \cdots c_l)$. A c_i ($= a_i \backslash b_i$) is obtained using the following table:

a_i \ b_i	$-\beta^l$	0	1
0	0	0	$-\alpha$
1	0	0	0

Here, α represents the total number of places where (0, 1) pair in (F_k, F^j) occurs. For example, consider F_k as (0 1 0 1 0 0 1), and F^j as ($1 - 2^1 - 2^1 - 2^2$ 1 $- 2^2$ 0). The COM1 operation obtains E_k as given below:

F_k	0	1	0	1	0	0	1
F^j	1	-2^1	-2^1	-2^2	1	-2^2	0
E_k	-2^1	0	0	0	-2^1	0	0

Note, $\alpha = 2$ as two (0, 1) pairs are present. On the other hand, the (\ ) operation with $F^j(1 - 2^1 - 2^2 - 2^1$ 1 $- 2^2$ 0) is shown to be null (ϕ).

F_k	0	1	0	1	0	0	1
F^j	1	-2^1	-2^2	-2^1	1	-2^2	0
E_k	null set (ϕ)						

For proposition P II, the COMpare (\ ) operator requires two cubes $F_k = (a_1\, a_2 \cdots a_l)$ and $F_j = (b_1\, b_2 \cdots b_l)$; where both $a_i, b_i \in \{0, 1\}$. Here, COM2 is defined in a straightforward manner as : $E_k' = F_k \backslash F_j = F_k\ pl\ F_j\ df\ F_k$, where "$pl$" and "$df$" are set operators. When the operands $F_k, F_j \in \{0, 1\}$, the

operator "*pl*" and "*df*" are bitwise OR and EOR, respectively. Let α be the total number of 1's present with E_k'. Replace 1's in E_k by $-\alpha^1$ to generate the conditional cube E_k. For example, consider "*pl*" and "*df*" operations for F_k and F_j given below:

F_k	0	1	0	1	0	1	0
F_j	0	1	0	0	1	0	1
	0	1	0	1	1	1	1
F_k	0	1	0	1	0	1	0
E_k'	0	0	0	0	1	0	1

; use "*pl*" (bitwise OR) operation

; use "*df*" (bitwise EOR) operation

The conditional cube E_k is, then, obtained from E_k' by replacing 1 by -2^1 in the positions of 1. Thus, $E_k = 0000 - 2^10 - 2^1$. Note, the COM1 operator detects and eliminates some of the redundant terms while other redundancies are deleted by the RED operator. The COM2 operator does not check redundancy. Nonetheless, COM2 operator is bit implementable, and offers a great advantage over COM1 from the aspects of computer memory and speed.

2) RED (/) Operator: Consider two conditional cubes $E_j = (\ c_1\ c_2\ \cdots\ c_l\ )$, and $E_k = (\ d_1\ d_2\ \cdots\ d_l\ )$, where $c_i, d_i \in \{-\beta^1, 0\}$. For $\alpha, \delta \in \beta^1$, the RED (/) operation is given by

$$E_j/E_k = \begin{cases} E_j; & \text{if } c_i = -\alpha \text{ in all those } \alpha \text{ positions} \\ & \text{where } d_i = -\delta \text{ and } \delta > \alpha \\ E_k; & \text{if } d_i = -\delta \text{ in all those } \delta \text{ positions} \\ & \text{where } c_i = -\alpha \text{ and } \alpha > \delta \\ \text{Retain } E_j, E_k; & \text{otherwise.} \end{cases}$$

Note, by REDucing either E_j or E_k, we remove redundant product terms. The remaining nonredundant E_i's are, henceforth said to form a minimal conditional cube (MCC). The following examples illustrate RED operation:

(i) E_j	0	-2^1	0	-2^1	0	(ii) E_j	0	-2^1	0	-2^1	0
E_k	0	-3^2	-3^2	-3^2	0	E_k	-3^2	-3^2	-3^2	0	0
E_j	0	-2^1	0	-2^1	0	Retain E_j and E_k					

For (i), $-\alpha = -2^1, -\delta = -3^2$, and $\delta > \alpha$. Moreover, -2^1's are in both positions where -3^2's are present. Thus, E_k is redundant and the result is E_j. A similar explanation follows for (ii). The RED operator, defined above, is suitable for P I. For notational simplicity, let us call it RED1. Following proposition P II, a simpler version (RED2) is adopted. Note, the COM2 operation generates subcubes E_k' which contains 0's and 1's only. Using E_k''s, and the concept explained in Section II-A, the redundancy checking required in RED2 operator is brought down to set theoretic operations "*pl*" and "*df*." This observation will help make RED2 implementation faster as compared with that for RED1. Using RED2, the examples (i) and (ii) can be solved as

(i) E_j'	0	1	0	1	0		
E_k'	0	1	1	1	0	; "*pl*" (OR operation)	
	0	1	1	1	0		
E_k'	0	1	1	1	0	; "*df*" (EOR operation)	
	0	0	0	0	0	; means E_j' is redundant	
(ii) E_j'	0	1	0	1	0		
E_k'	1	1	1	0	0	; "*pl*" (OR operation)	
	1	1	1	1	0		
E_k'	1	1	1	0	0	; "*df*" (EOR operation)	
E_k'	0	0	0	1	0	; nonnull means retain both cubes	

3) CMB (*) Operator: The CoMBine (*) operator processes MCC E_i's as its operands. Before applying CMB, partition the set of E_i's into independent (IG) and dependent (DG) groups. The MCC's that belong to IG are already mutually disjoint among themselves. Thus, generating the disjoint events F^1 from IG is straightforward. It has been observed in [1] that most (s, t) paths in large DCS networks (which are loosely connected type) do not have common elements among them. The partitioning of E_i's into IG and DG can be embedded in the implementation of RED (/) operation (refer to Section IV-B), which avoids an unnecessary taxing of the implementation of CMB (*) operator. Moreover, the processing cost (as will be clear later in this section) for IG is far less than that for DG, the overall improvement in the performance of the algorithm is obvious.

Definition: Consider MCC's E_j and E_k whose elements $c_i, d_i \in \{-\beta^1, 0\}$. For $1 \le i \le l$, if there exists at least one (c_i, d_i) pair for $c_i, d_i \neq 0$, the E_j and E_k are said to form dependent group (DG). As an example consider E_j as $(-2^10 - 2^1\ 0\ 0\ 0\ 0)$, and E_k as $(-3^200 - 3^2 - 3^2\ 0\ 0)$. Here, E_j and E_k belong to DG, since they have a common element in position 1. Use this definition to select out DG's from nonredundant MCC E_i's. The remaining E_i terms form IG's. The (E_j, E_k) entry in IG has no common elements among themselves. It means the (c_i, d_i) pair will always be of the types $(0, 0), (-\alpha, 0)$, and $(0, -\delta)$. Considering this, terms like $(0 - 2^3000 - 2^3\ 0)$ and $(0\ 0\ 0\ 0\ 0\ 0\ -1)$ belong to IG. Note, these terms are independent with both E_j and E_k considered above. For notational simplicity, we denote the elements of independent (dependent) group by IG_i (DG_i). For $|\ E_i\ | = r$, $|\ IG_i\ | = \eta$, and $|\ DG_i\ | = \nu, \eta + \nu = r$. Note, an element E_i belongs to either IG or DG. The CMB operator differentiates our algorithm CAREL with [1], [12], [16], [19]. We discuss the differences in Section V. The following two cases define CMB: CASE 1 (CASE 2) is used for independent (dependent) group.

CASE 1 (CMB for independent group): For $1 \le j < \eta$, the CMB operates iteratively as

$$IG_{j+1} = IG_j * IG_{j+1} \tag{6}$$

where "*" is a "*pl*" operation such that $0\ pl\ 0 = 0$, $0\ pl - \delta = -\delta$, and $-\alpha\ pl\ 0 = -\alpha$. Equation (6) states that we will eventually get one term IG_η.

Example: Consider four IG_i's, namely, $IG_1(00 - 2^1 - 2^10000)$, IG_2 (-10 000 000)), $IG_3(0 - 3^200 - 3^20 - 3^20)$, and $IG_4(00000 - 2^40 - 2^4)$. We get IG_η as follows:

IG_1	0	0	-2^1	-2^1	0	0	0	0
IG_2	-1	0	0	0	0	0	0	0
IG_3	-1	0	-2^1	-2^1	0	0	0	0

TG_1	0	-3^1	0	0	-3^1	0	-3^1	0
TG_2	-1	-3^1	-2^1	-2^1	-3^1	0	-3^1	0
TG_3	0	0	0	0	0	-2^4	0	-2^4
TG_4	-1	-3^1	-2^1	-2^1	-3^1	-2^4	-3^1	-2^4

CASE 2 (CMB for dependent group): In this case, the CMB operator is quite involved. To define this operator, use Steps 1 and 2 below:

Step 1: $DG_1 * DG_2$ generates two cubes TG_1 and TG_2. The cube TG_1 has $-\theta$ in those θ positions where cubes DG_1 and DG_2 both are negative, while others are "0"s. TG_2 has "1" in all θ positions and its remaining entries are generated from DG_1 and DG_2 using $0\ pl\ x = x\ pl\ 0 = x; x \in \{-3^1, 0\}$. Before applying "$pl$" operation, update variable x by adding θ to x. Note, the entries in TG's belong to $\{-3^1, 0, 1\}$.

Step 2: Consider $TG_1 = (f_1\ f_2 \cdots f_l)$ and $DG_1 = (d_1\ d_2 \cdots d_l)$; where $f_i \in \{-\alpha^1, 0, 1\}, d_i \in \{-\delta^1, 0\}$ and $\alpha^1, \delta^1 \in \{3^1\}$. The following substeps obtains TG_1's.

```
for (all DG_i) begin /* i = 3,....*/
    for (all TG_j) begin /* j = 1,....*/
      call DG (TG_j, DG_i, TG'); /* TG' is the result */
      if (terminate)
        j = j + 1;
    end;
    TG = TG'; /*we create new TG_j list */
end;
```

Step 2 needs a procedure **DG** ( ) to obtain various TG_i's. An algorithm for DG(TG_j, DG_i, TG') is as follows. Note, to make the algorithm easier to follow, we provide an example for each case (a)–(f) in Boolean expression where X is any Boolean expression.

```
while (TRUE)begin /* forever, stop when terminate=TRUE */
    delta = Number of (1,-δ^1) pairs;
    alpha = Number of (-α^1, -δ^1) pairs;
    /* Consider the following cases of (delta, alpha) */
    (delta == δ^1,alpha == don't care) : /* case a) */
     begin /* (abcX)(abc) = φ */
      terminate = TRUE; return;
     end;
    (delta > 0, alpha == don't care) : /* case b) */
    begin /* (abX)(abcde) = (abX)(cde) */
      for (k =1 to l) begin
        if (DG_i[k] == -δ^1) begin
          if(TG_j[k] == 1)
            DG_i[k] = 0;
          else
            DG_i[k] = DG_i[k] + delta;
        end;
      end;
    end;
    (δ == 0, α == 0) : /* case c) */
    begin /* CMB for independent group */
      TG_j' = TG_j pl DG_i; /* (abX)(cd)=abcdX */
      append TG_j' to TG';
      terminate = TRUE; return;
    end;
    (delta == 0, alpha == α^1) : /* case d) */
    begin /* (abc) (abX) = abX */
      append TG_j to TG';
      terminate = TRUE; return;
    end;
    (delta == 0, alpha == δ^1) : /* case e) */
    begin * (aoc.X) (ao) = ab.X */
      TG_j = DG_i pl (the rest of elements in TG_j other than -α^1);
      append TG_j' to TG';
      terminate = TRUE; return;
    end;
    OTHERWISE : /* case f) */
    * (abe.X) (abcd) = abX + (abe.X)(cd) */
    * note, the first term above is one of the final result */
    begin
      for(k = 1 to l) begin
        if(DG_i[k] == -δ^1) begin
          if(TG_j[k] == -α^1) begin
            TG_j'[k] = -alpha;
            TG_i[k] = 1;
            DG_i[k] = 0;
          end;
          else begin
            TG_j'[k] = TG_j[k];
            DG_i[k] = DG_i[k] + alpha - delta;
          end;
        end;
        else if(TG_j[k] == -α^1) begin
          TG_i[k] = 0;
          TG_j[k] = TG_j[k] + alpha;
        end;
        else
          TG_j'[k] = TG_j[k];
      end;
      append TG_j' to TG';
    end;
end;
```

Note, cases a)–f) are obtained from Theorems A.1 and A.2 in the Appendix. A proof on completeness of CMB operator is also given in the Appendix.

Example: Assume $DG_1(-3^10-3^1-3^1000000)$, $DG_2(-4^20-4^20-4^20000-4^2)$, $DG_3(-3^30-3^300-3^30000)$ and $DG_4(-4^400-4^40-4^4-4^4000)$, where $DG_i \in$ DG for $1 \le i \le 4$.

Step 1. We generate two cubes TG_1 and TG_2 as follows:

DG_1	-3^1	0	-3^1	-3^1	0	0	0	0	0	0
DG_2	-4^2	0	-4^2	0	-4^2	0	0	0	0	-4^2
TG_1	-2^1	0	-2^1	0	0	0	0	0	0	0
TG_2	1	0	1	-1	-2^2	0	0	0	0	-2^2

Step 2. Consider $TG_1 * DG_3$ and $TG_2 * DG_3$. Using procedure **DG** ( ), $TG_1' = TG_1$ because $TG_1 * DG_3$ is of case (d) (refer to the procedure). The $TG_2 * DG_3$ is computed as follows:

TG_2	1	0	1	-1	-2^2	0	0	0	0	-2^2	
DG_3	-3^3	0	-3^3	0	0	-3^3	0	0	0	0	; case b) delta=2
TG_2	1	0	1	-1	-2^2	0	0	0	0	-2^2	; keep TG_2
DG_3	0	0	0	0	0	-1	0	0	0	0	; update DG_3
TG_2'	1	0	1	-1	-2^2	-1	0	0	0	-2^2	; case c)

Thus, the new TG_i's are: $TG_1(-2^10 - 2^10000000)$ and $TG_2(101 - 1 - 2^2 - 1000 - 2^2)$. They are further CoMBine with DG_4.

$TG_1 * DG_4$:

TG_1	-2^1	0	-2^1	0	0	0	0	0	0	0	
DG_4	-1^4	0	0	-1^4	0	-1^4	-1^4	0	0	0	; case f)
TG_1'	-1	0	0	0	0	0	0	0	0	0	; $TG_1 * DG_4$ obtains
TG_2'	1	0	-1	-3^4	0	-3^4	-3^4	0	0	0	; TG_1' and TG_2'

$TG_2 * DG_4$:

TG_2	1	0	1	-1	-2^2	-1	0	0	0	-2^2	
DG_4	-1^4	0	0	-1^4	0	-1^4	-1^4	0	0	0	; case b)
TG_2	1	0	1	-1	-2^2	-1	0	0	0	-2^2	; keep TG_2
DG_4	0	0	0	-3^4	0	-3^4	-3^4	0	0	0	; case d)
TG_3'	1	0	1	-1	-2^2	-1	0	0	0	-2^2	; the result

We get three TG_i's : TG_1 (-1000000000), TG_2 ($10-1-3^40-3^4-3^4000$), and TG_3 ($101-1-2^2-1000-2^2$). To help understand these steps, we provide a step by step computation using Boolean notation. The four DG_i's are equivalent to $\overline{acd}$, $\overline{ace}j$, $\overline{acf}$, and $\overline{aafg}$. We have assumed that a DG_i is a function of "a" through "j" Boolean variables. The CMB operator determines

$$(\overline{acd})(\overline{ace}j)(\overline{acf})(\overline{adfg}) = (\overline{ac} + ac\bar{d}\,\overline{ej})(\overline{acf})(\overline{adfg})$$
$$= (\overline{ac}\,\overline{acf} + ac\bar{d}\,\overline{ej}\,\overline{acf})(\overline{adfg}) = (\overline{ac} + (ac\bar{d}\,\overline{ej})(\bar{f}))(\overline{adfg})$$
$$= (\overline{ac} + ac\bar{d}\,\overline{ej}\,\bar{f})(\overline{aafg}) = (\overline{ac}\,\overline{adfg} + ac\bar{d}\,\overline{ej}\,\bar{f}\,\overline{adfg})$$
$$= \bar{a} + a\bar{c}\,\overline{dfg} + (ac\bar{d}\,\overline{ej}\,\bar{f})(\overline{dfg}) = \bar{a} + a\bar{c}\,\overline{dfg} + ac\bar{d}\,\overline{ej}\,\bar{f}.$$

The above example is solved using following Boolean identities [20]:

$$(\bar{x}+\bar{y})(\bar{x}+\bar{z}) = \bar{x} + \bar{y}\,\bar{z}\,;\ \bar{x}\cdot\overline{\bar{x}\bar{y}} = \bar{x};$$
$$\overline{\bar{x}\bar{y}} = (\bar{x} + x\bar{y}).$$

4) GEN ($\ominus$) Operator: Consider the path identifier F_j ($a_1\ a_2\ \cdots\ a_l$), the cubes generated from CASE 1 and CASE 2 CMB operation as IG_n ($e_1\ e_2\ \cdots\ e_l$), and TG_k ($f_1\ f_2\ \cdots\ f_l$); where $a_i \in \{0, 1\}$, $e_i \in \{-\beta^l, 0\}$, and $f_i \in \{-\beta^l, 0, 1\}$. The GEN ($\ominus$) operator, then, obtains F^j's: $F^j = F_j \ominus IG_n \ominus TG_k$. As an example, assume cubes F_j, IG_n, and TG_k. Then F^j is computed as

F_j	0	0	1	1	0	0	0	0	0	0
IG_n	-2^1	-2^1	0	0	0	0	-1	0	0	0
TG_k	0	0	0	0	1	-2^2	0	-2^2	1	0
F^j	-2^1	-2^1	1	1	1	-2^2	-1	-2^2	1	0

For a $b_i \in F^j$, obtain $b_i = a_i \oplus e_i \oplus f_i$. The bitwise operation $\oplus$ is shown below:

$$0\oplus 0\oplus 0 = 0$$
$$1\oplus 0\oplus 0 = 1$$
$$0\oplus -\beta^l \oplus 0 = -\beta^l$$
$$0\oplus 0\oplus 1 = 1$$
$$0\oplus 0\ominus -\beta^l = -\beta^l.$$

Here, we have considered only those five (out of 12) combinations which are feasible.

Theorem 1: The combinations $(0, 1, 0)$ and $(-\beta^l, 0)$ are not possible.

Proof: Using the definition of IG_n, it is easy to show that the occurrence of $(0, 1, 0)$ is not possible. Moreover, F_j represents a path identifier defined over $\{0, 1\}$. Thus, a "$-\beta^l$" entry in F_j does not appear with F_j. □

Theorem 2: More than one occurrence of 1 or $-\beta^l$ in position i for the cubes (F_j, IG_n, TG_k) is not feasible.

Proof: A multiple occurrence of 1 in position i confirms the presence of an uncomplemented variable with (F_j, TG_k) combination. The cubes IG_n and TG_k represent disjoint expression, and are generated for a path identifier F_j. Thus, the possibility of having two 1's in position i does not arise. A similar argument follows for multiple $-\beta^l$ or $(1, -\beta^l)$ combinations. □

IV. CAREL: Algorithm and Implementation

A. Algorithm

The steps of the proposed algorithm are shown below.

CAREL:
begin
  Sort F_i in ascending cardinality, and for terms /*refer to [12], [15] for its advantages*/
  of the same size, use lexicographic ordering;
  $F^1 = F_1$;
  for all paths F_i **begin** /*$i = 2, \cdots$ */
    COM (F_i,F_j) ; /*$j = 1, \cdots, i-1$; we get E_i's */
    RED (E_i,E_j) ; /* return irredundant IG's and DG's */
    CMB (IG_i, DG_i) ; /* produce IG_n and TG_i */
    GEN (F_i, IG_n, TG_i) ; /* get F^j's */
  end;
  Compute the reliability(unreliability) value $R(G)$ ($Q(G)$);
end.

Consider COM1 and RED1 (COM2 and RED2) operations while using CAREL with P I (P II) propositions. Thus, CAREL applies equally for P I and P II. To compute the terminal reliability (unreliability) parameter, we have developed a program called *re_num* which accepts the output from GEN (F_i, IG_n, TG_i). For given value of link reliability, *re_num* produces a numerical value for the terminal reliability. One may use any other software package like *vaxima* [18] to evaluate the reliability or unreliability expression. Tables III–V show the reliability figures for 19 different types of DCS networks.

B. Implementation

This section describes an implementation of CAREL for proposition P II. It is based on bit operations. The implementation of CAREL P I follows similarly, and will not be discussed. Both CAREL P I and P II are written in C. In what follows, we consider four macros which define bit operations. The cost for each macro call is also given.

1) SetUnion ($s1$, $s2$, s) ; /* $s = s1 \cup s2$ */ cost : $\lceil l/16 \rceil$ *assignment* operations.
2) SetDif ($s1$, $s2$, s) ; /* $s = s1 - s2$ */ cost : $\lceil l/16 \rceil$ *EOR* operations.
3) SetCompare ($s1$, $s2$) ; /* return TRUE if $s1 == s2$ */ cost : $\leq \lceil l/16 \rceil$ *if* statements.
4) SubSet ($s1$, $s2$) ; /* return TRUE if $s1 \subseteq s2$ */
   cost : 1 SetUnion ( ) + 1 SetCompare ( ) calls.

In 1)–4), l represents the number of links of the network and a word "w" is of 16 bits. These four bit operations are used to implement CAREL operators.

1) COM ( ) Implementation: A procedure COM (F_i, F_j) is implemented as follows

```
for (j = 1 to i − 1) begin
    SetUnion (F_i, F_j, s); /* s is temporary result */
    SetDif (s, F_j, E_i'); /* E_i' is the result,append it to E_i' list */
end;
```

Note, the number of COM ( ) function calls is $1 - 2 - \cdots - (n-1) = \frac{n(n-1)}{2}$ times in a network with n paths (cuts).

2) RED ( ) Implementation: In COM ( ) operation, we produce E_k (refer to Section III-B1). The E_k' and bit operations obtain non redundant MCC's. The implementation of RED operator is shown below:

```
for all E_i''s begin /* i=1, ··· */
    if (SubSet(E_i', E_k') ) /* E_k' is REDucible by E_i' */
        dispose E_k'; break;
    else if (SubSet(E_k', E_i') ) /* E_i' is REDucible by E_k' */
        dispose E_i';
    else
        create IG and DG groups;
end;
if E_k' was not disposed
    add E_k' to E_i' list;
```

Note, the nonredundant E_i''s are in bit form (having 0's and 1's only). The MCC E_i's are generated from E_i''s by replacing 1 by $-\alpha^l$ in the positions of 1, where $\alpha \in \beta^l$ (refer to Section III-B). The number of RED ( ) function calls is n times in a network with n paths (cuts). The number of loopings inside the RED ( ) function depends on the network type, and also on the path identifier F_i for which it is called. For the worst case, RED( ) for F_i needs i loopings, and hence the computation of RED operator is of the order $O(n^2)$.

3) CMB (*) Implementation: The CMB (*) operator is the most time consuming operator out of all the four operators we have used in our method. The implementation of CMB for independent group is straightforward, and is shown first:

```
for (all IG_i) begin /* i = 1, ··· η − 1 */
    for (j = 1 to l) begin
        if(IG_i[j] ≠ 0)
            IG_{i+1}[j] = IG_i[j];
    end;
end;
```

The implementation of CMB for dependent group is expensive. Note, a DG_i contains $(-\beta^l, 0, 1)$; we cannot utilize bit operations for CMB operator. To update the contents of the CMB operands, DG_i and TG_j, as well as to generate TG_j', we trace the contents of the operands element by element. The computation time is highly data dependent. But, in any case (refer to cases (a) through (f) in the procedure considered in Section III-B3), the order of computation time is $O(l)$. Note, in our program, Step 1 of the algorithm falls into case (f). The maximum number of DG_i's is k for generating *e.m.d.* event(s) for F_k, and the number of generated TG_i is $O(k)$. Hence, the worst case cost for calling CMB( ) is $O(k^2)$ of **DG** ( ) calls.

4) GEN (⊕) Implementation: This operator is processed by sequentially tracing the contents of F_i and IG_n for generating F^n(s) events. The procedure GEN(F_i, IG_n, TG_k) implements ⊕ operator:

```
for (all TG_k's) begin /*k = 1, ···*/
    for (j = 1 to l) begin
        if (F_i[j] ≠ 0)
            F^n[j] = F_i[j];
        else if (IG_n[j] ≠ 0)
            F^n[j] = IG_n[j];
        else
            F^n[j] = TG_k[j];
    end;
end;
```

Note, for a path identifier F_i, we get disjoint event(s) F^n. The time complexity involved in GEN ( ) for a network depends on the number of the generated *e.m.d.* events F^n's. If the maximum number of *e.m.d.* events is m, the worst case complexity is of the order $O(mn)$.

C. Illustrating Examples

Example: Consider Fig. 1 with (s, t) paths ab, cd, ade, and bce. Paths are encoded as path identifiers F_i's ($1 \le i \le 4$), and are sorted in their ascending cardinality as:

F_1	0000000000000011	F_3	0000000000011001
F_2	0000000000001100	F_4	0000000000010110.

Following the algorithm given in Section IV-A, $F^1 = F_1$. To generate *e.m.d.* event(s) for F_3, we start with COM(F_3, F_1) and COM(F_3, F_2) which give E_1'(0000000000000010) and E_2' (0000000000000100) respectively. RED(E_i) retains both E''s, and groups them into IG with an empty DG group. CMB operation obtains IG_n as 0000000000000-1-10. Finally, GEN(F_3, IG_n, DG) gives F^3 as (0000000000011-1-11), which can be interpreted in an intermediate form as $ade(1-b)(1-c)$. Note, this resultant expression has one to one correspondent with the probability expression. Similarly, we obtain F^2 and F^4. The various *e.m.d.* events are:

F(disjoint) = $ab + (1-ab)cd + ade(1-b)(1-c) + bce(1-a)(1-d)$

Example: Consider Fig. 4 and its 13 path identifiers as

F_1 (0000000100100010)	F_2 (0000000100010001)
F_3 (0000000010001001)	F_4 (0000000011100010)
F_5 (0000000010001110)	F_6 (0000000100010110)
F_7 (0000000011010001)	F_8 (0000000100100101)
F_9 (0000000101001001)	F_{10} (0000000101001110)
F_{11} (0000000011010110)	F_{12} (0000000010111010)
F_{13} (0000000011100101).	

The *e.m.d.* event(s) for F_8 is generated using the steps mentioned in the algorithm. The details are shown below: For $1 \le i \le 7$, COM(F_8,F_i) obtains

E_1' (0000000000000010), E_2' (0000000000010000),
E_3' (0000000010001000), E_4' (0000000011000010),
E_5' (0000000010001010), E_6' (0000000000010010),
and E_7' (0000000011010000).

The RED operation, then, removes E_4' through E_7', because these terms are redundant with respect to either E_1' or E_2'. The RED also classifies the nonredundant terms into IG and an empty DG set. The CMB operator obtains IG_n as 0000000−$2^1$00-1−$2^1$0-10. Finally GEN operator generates F^8 as (00000001−$2^1$01-1−$2^1$1-11).

Example: This example shows the generation of *e.m.d.* event(s) for F_9 and, thus, illustrates the concept of DG's. For $1 \le i \le 8$, COM(F_9,F_i) obtains

E_1' (0000000000100010), E_2' (0000000000010000),
E_3' (0000000010000000), E_4' (0000000010100010),
E_5' (0000000010000110), E_6' (00 0000000010110),
E_7' (0000000010010000), and E_8' (0)000000000100100).

E_4' through E_7' are REDuced. The RED procedure lists E_1' and E_3' with IG, while E_1' and E_8' are kept in DG set. The CMB operation obtains IG_n as 00000000-100-10000. The elements in

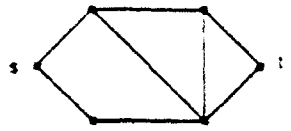


Fig. 2. 6-node, 8-link network.

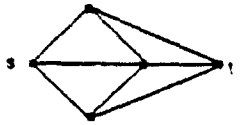


Fig. 3. 5-node, 8-link Network

DG group are used to generate TG_1 and TG_2 (in step 1) as follows:

DG_1	0	0	0	0	0	0	0	0	0	0	-2^1	0	0	0	-2^1	0
DG_2	0	0	0	0	0	0	0	0	0	0	-2^2	0	0	-2^2	0	0
TG_1	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0
TG_2	0	0	0	0	0	0	0	0	0	0	1	0	0	-1	-1	0

Since we do not have any other DG_i's, Step 2 of CMB operation is skipped. Terms TG_1 and TG_2 are the final results for CASE 2 (of CMB). Hence, from GEN operator, we get two *e.m.d.* events for F_9, namely (00000001-11-1-11001) and (00000001-111-11-1-11). An expression for F(disjoint) of Fig. 4 is given as

$$
\begin{aligned}
&bfi + aei(1-bf) + adh((1-i) + i(1-bf)(i-e)) \\
&+ bfgh(1-ad)(1-i). + bcdh(1-a)((1-f) \\
&+ f(1-i)(1-g)) + bcei(1-a)(1-dh)(1-f) \\
&+ aegh(1-bf)(1-d)(1-i) + acfi(1-b)(1-dh)(1-e) \\
&+ adgi((1-e)(1-f)(1-h) \\
&+ (1-b)(1-c)(1-e)f(1-h)) \\
&+ bcdgi(1-a)(1-e)(1-f)(1-h) \\
&+ bcegh(1-a)(1-d)(1-f)(1-i) \\
&+ bdefh(1-a)(1-c)(1-g)(1-i) \\
&+ acfgh(1-b)(1-d)(1-e)(1-i).
\end{aligned}
$$

Assuming that each link has 0.9 as probability of survivability, the terminal reliability for the DCS (Fig. 4) is 0.977814.

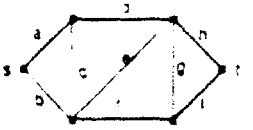


Fig. 4. Modified ARPANET [3].

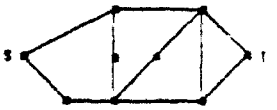


Fig. 5. ARPANET in 1971.

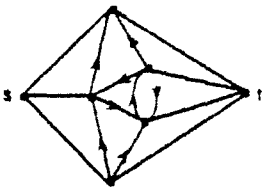


Fig. 6. 7-node, 15-link network.

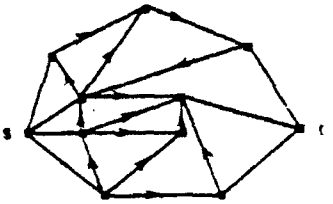


Fig. 7. 11-node, 21-link network.

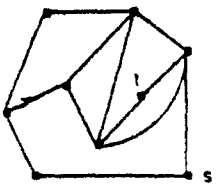


Fig. 8. 9-node, 13-link network.

V. Experimental Results and Discussion

Table II gives us the information of the networks used in our experiments. We name a network $N^{\#cuts}_{\#paths}$, where # paths and # cuts are the number of minpaths and mincuts, respectively. As shown, Figs. 1–8 possesses less than 20 paths, while there are 281 and 780 paths with the DCS given in Figs. 17 and 18, and 19, respectively. Tables III–V provide results for 19 different types of DCS networks. We have considered two performance parameters as

i) The number of disjoint terms generated (DPath), and

ii) The computer time involved to obtain reliability figure. This includes processing time for i) and also the reliability or unreliability expression evaluation time using *re_num* or *vaxima* [18].

First, the performance (refer to Table III) of CAREL (with P I option) is compared with that of VT [18], a representative method for proposition P III. The reliability values obtained in both CAREL and VT [18] are exactly the same. CAREL (with P I option) obtains the same order of *e.m.d.* events as in [18]. The number of *e.m.d.* events generated is influenced by the ordering of the minpaths/mincuts [13]. When we scrambled the ordering of the paths, we got more (less) number of events than the ones reported in the table with exactly the same reliability values. We notice that when a network generates less number of *e.m.d.* events, it requires less computation time. From this observation, we believe that preprocessed minpaths/mincuts (in addition to

TABLE II
NETWORKS USED IN EXPERIMENTS

Network	Paths	Cuts	Comments
N_1^4	4	4	Fig. 1 Bridge network
N_2^7	7	9	Fig. 2 5-node, 8-link network
N_3^9	9	8	Fig. 3 5-node, 8-link network
N_3^{13}	13	9	Fig. 4 Modified ARPANET [3]
N_3^{28}	13	28	Fig. 5 ARPANET in 1971
N_4^{18}	14	18	Fig. 6 7-node, 15-link network
N_5^{110}	18	110	Fig. 7 11-node, 21-link network
N_5^{24}	18	24	Fig. 8 9-node, 13-link network
N_{24}^{19}	24	19	Fig. 9 8-node, 12-link network [3]
N_{20}^{20}	20	20	Fig. 10 Fig. 9 with different source
N_{12}^{20}	25	20	Fig. 11 7-node, 12-link network
N_{29}^{29}	29	29	Fig. 12 8-node, 13-link network [18]
N_{16}^{396}	36	396	Fig. 13 16-node, 30-link network [14]
N_{14}^{528}	44	528	Fig. 14 ARPANET [18]
N_{14}^{25}	44	25	Fig. 15 Reduced form of Fig. 14 [18]
N_{64}^{78}	64	78	Fig. 16 10-node, 21-link [18]
N_{281}^{1300}	281	1300	Fig. 17 ARPANET [14]
N_{281}^{214}	281	214	Fig. 18 Reduced form of Fig.17 ARPANET
N_{30}^{7376}	780	7376	Fig. 19 20-node, 30-link [10]

sorting them in ascending cardinality as suggested in [12]), is needed to further improve the performances of existing methods. The results in Table III show the superiority of CAREL (with P I option) as compared to VT [18], a *representative technique* of proposition P III. The running time improvement in CAREL is more noticeable when we evaluate larger networks (Figs. 16 and 18). The results support the discussion in Section II-C, and analysis in [1]. Note, COM1 and RED1 (COM2 and RED2) is used in CAREL with P I (P II) option. Thus, CAREL P I and CAREL P II differ only in generating the minimal conditional set (MCC) E_i's. Like other methods in proposition P II, CAREL P II obtains the MCC's from minpath/mincut of a network (4), while CAREL P I incorporates (2) (Section II-B). The COM1, in addition to generating conditional set, reduces some of the redundant terms too. Thus, RED1 operation in CAREL P I utilizes less number of operands than that used in RED2. However, COM1 operator utilizes more terms than COM2 (refer to Section II-C). Bit operations are used to implement CAREL P II and is one additional factor which makes this method run faster than CAREL P I. The other problem with CAREL P I method is on the memory size used. To incorporate (2), this method has to maintain a list of *e.m.d.* events. For a large network which generates more than 50 000 events (Table IV), the program requires a huge memory space that is available only on large system. Furthermore, the huge number of events reduces the speed of CAREL P I significantly. CAREL P II, as well as CAREL P I, requires a list of minpath/mincut. Since one path / cut in a network which contains l links needs $\lceil l/w \rceil$ word for w word size, this requirement does not prevent CAREL P II of solving large distributed system. Both variants of CAREL use the same CMB and GEN operators for their OP_i and OP. In CAREL P II, generating *e.m.d.* event(s) of a path identifier F_i is independent from other *e.m.d.* events obtained for other path identifiers F_j's. From this observation, we notice that minor changes in CAREL P II will make it quite suitable for parallel system implementation. Overall, CAREL P II is a better method compared to CAREL P I.

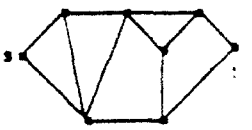


Fig. 9. 8-node, 12-link network [3].

Table IV shows the comparisons of CAREL P I and CAREL II in term of *e.m.d.* events generated, the reliability values, a

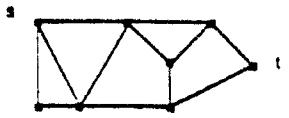


Fig. 10. 8-node, 12-link network (Fig. 9) with different source.

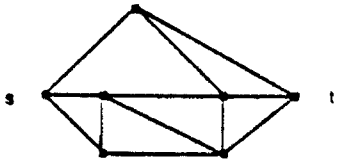


Fig. 11. 7-node, 12-link network.

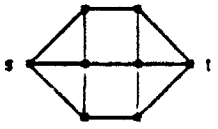


Fig. 12. 8-node, 13-link network [18].

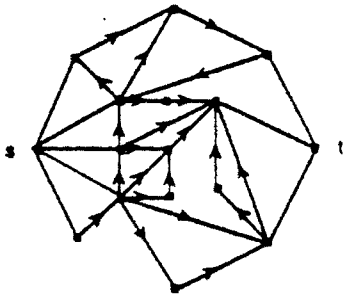


Fig. 13. 16-node, 30-link network [14].

Fig. 14. ARPANET [18].

the running time to obtain these values. Both methods get the same *e.m.d.* events and reliability values. However, CAREL P II obtains the results faster than CAREL P I as we can see from the table. Furthermore, as expected from our previous discussion, CAREL P I has problem evaluating large networks (e.g., Fig. 19). The results show that proposition P II is better than P I, which back up our previous discussion in Section II-C. We, thus, conclude that CAREL P II (implementation of proposition P II)

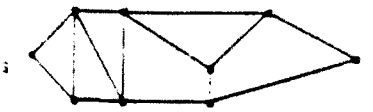


Fig. 15. ARPANET (Fig. 14) after series–parallel reductions

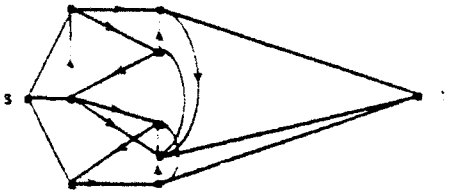


Fig. 16. 10-node, 21-link [18].

is a better method compared to both CAREL P I (proposition P I) and VT [18] (proposition P III).

CAREL P II is more efficient compared with SYREL [1]. SYREL needs 0.8 s (on VAX 11/750) to get the terminal reliability value for network shown in Fig. 9, while CAREL P II uses only 0.1 s (on Encore Multimax). CAREL P II, while keeping the algorithm efficient, produces less number of disjoint events expressions compared to SYREL. Our CAREL cube notation enables the algorithm to produce concise expression, hence less number of events that reduce the running time of the algorithm. For example, SYREL produces disjoint events F^3 as adh ($\bar{i} + \bar{b}\bar{e}i + \bar{f}\bar{e}$ bi), while CAREL generates adh ($\bar{i} - \bar{e}\bar{b}\bar{f}$ i), which contain 3 and 2 terms, respectively. We do not provide other comparison results of CAREL and SYREL [1] because SYREL does not report any results for larger sized networks. Since CAREL P II basically combines the best feature of SYREL [1] (i.e., using MCC) and take into account the advantage of using variable groupings of [3], we expect CAREL P II to outperform both SYREL [1] and [3] as verified by our experimental results.

The comparisons of evaluating reliability and unreliability values of distributed system networks are shown in Table V. We use the same networks as in Table II, and utilize CAREL with P II option. The unreliability values are obtained from the *e.m.d.* events of mincuts of the networks. We generate mincuts from minpaths using a simple program based on a method discussed in [2]. However, one may use other methods for this step. In most DCS networks (shown here), we obtain more number of cuts than that of paths. Thus, evaluating the reliability values is faster than computing the unreliability values. For networks which have less number of cuts than paths, the opposite is true. The table shows that the sum of the reliability and unreliability figure of a network is always 1 (as expected); however, rounding-off of 10 000 or more terms produces a little quantization error for large networks. The values obtained further support the correctness of our proposed algorithm. Hence, reliability (unreliability) value of a network can be obtained from the computed unreliability (reliability). The results show that CAREL is capable of evaluating both reliability and unreliability values of large distributed system networks.

VI. Conclusion

We have proposed an efficient algorithm called CAREL which computes the terminal reliability or unreliability of moderate to large sized DCS networks with modest memory and time

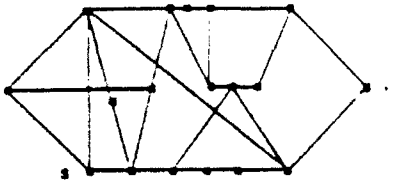


Fig. 17. ARPANET [14].

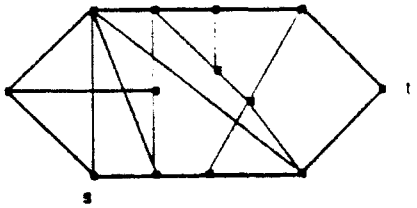


Fig. 18. Reduced form of Fig. 17.

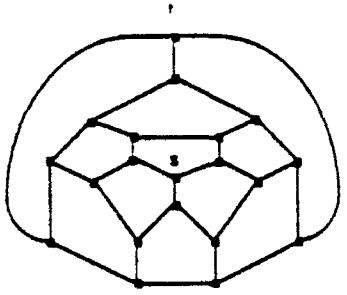


Fig. 19. 20-node, 30-link network [10].

requirements. The algorithm has been implemented in C and was run on an Encore MULTIMAX 320 system. The performance of a program is usually based on the algorithm, the data structure and the language used, the computer on which the program is run, and last but not the least, the coding of the program [8]. Since different programmers produce different codings for an algorithm, the human factor (in want of sufficient data) is inappropriate while comparing various techniques. A better implementation or faster machine would increase the performance of a program, but only to a factor of 10 [11]. Moreover, all methods of reliability computation are known to be computationally intractable or NP-hard, which makes difficult to compare the techniques from the aspect of complexity [2], [9], [10], [26], [27]. CAREL is faster than other existing Boolean algorithms. This is obvious from the CPU time requirement for solving the terminal reliability/unreliability of various DCS networks. Note, the CAREL combines the advantages offered both by SYREL [1] and the method given in [3]. Presently, we are utilizing CAREL to help compute the reliability issues of multiprocessor system [6], [28], [29], [30]. An earlier version of CAREL has successfully been applied to solve reliability problems in one type of redundant path MIN [19].

TABLE III
COMPARISON OF CAREL WITH VT [18] FOR CPU TIME AND DISJOINT PATHS

Network	VT[18][1]		CAREL P.D.[2]		RIG
	DPath	Time (s)	DPath	Time (s)	
N_1^4	***	***	4	0.0	0.978480
N_2^9	9	0.0	7	0.0	0.968425
N_9^{11}	11	0.0	11	0.1	0.997632
N_{13}^{16}	16	0.1	16	0.0	0.977184
N_{13}^{28}	16	0.2	15	0.1	0.964855
N_{14}^{18}	***	***	23	0.1	0.996665
N_{18}^{110}	82	3.1	94	0.2	0.994076
N_{18}^{24}	27	0.5	30	0.1	0.969112
N_{24}^{19}	41	1.2	39	0.1	0.975116
N_{20}^{20}	***	***	34	0.1	0.984068
N_{25}^{20}	***	***	50	0.1	0.997494
N_{29}^{29}	77	3.0	76	0.1	0.996217
N_{36}^{396}	467	74.2	442	0.6	0.997186
N_{44}^{328}	93	16.8	105	0.2	0.904577
N_{44}^{25}	90	9.1	87	0.2	0.974145
N_{64}^{78}	305	78.8	309	0.5	0.997506
N_{281}^{1300}	***	***	2491	4.2	0.985928
N_{281}^{214}	2085	8788.3	2386	3.6	0.987390

1) Run on a Convex C1-XP system.
2) Run on an Encore Multimax system.
Link Reliability = 0.9.
**** Results not known.
Time is in CPU seconds.

V. Appendix
Proving Correctness of CAREL

The CAREL uses four operators, namely COM, RED, CMB and GEN to transform an expression of paths (or cuts) into an equivalent exclusive and mutually disjoint (*e.m.d.*) expression. The COM operator defines conditional cubes E_i's for a F_i, while RED removes redundant E_i's to provide minimal conditional cube (MCC). The operation CMB combines MCC to generate disjointing terms (DT's). The DT's are mutually disjoint. Moreover, the F_i and its DT's (refer to GEN operator) form expression which is disjoint with all other terms in (1). In what follows, we discuss that the CAREL always generates *e.m.d.* terms (DT's) for an F_i.

Section III-B1 describes COM operator. A conditional cube E_i considers "α" elements of F_i which are not present in F_j. To represent E_i DOWN (refer to Section III-A), our notations use $-\alpha^l$ in the positions of the variable (here the index l is equal to j). Considering CAREL (Section IV-A), it is obvious that COM

TABLE IV
COMPARISON OF CAREL USING P I AND P II FOR CPU *time*[1] AND DISJOINT PATHS

Network	CAREL (P I)		CAREL (P II)		$R(G)$‡
	DPath	Time (s)	DPath	Time (s)	
V_4^4	4	0.0	4	0.0	0.978480
V_9^2	7	0.0	7	0.0	0.968425
V_9^5	11	0.0	11	0.0	0.997632
V_{13}^9	16	0.0	16	0.0	0.977184
V_{13}^{28}	15	0.1	15	0.0	0.964855
V_{14}^{18}	23	0.1	23	0.1	0.996665
V_{18}^{110}	94	0.2	94	0.2	0.994076
V_{18}^{24}	30	0.1	30	0.1	0.969112
V_{24}^{19}	39	0.1	39	0.1	0.975116
V_{20}^{20}	34	0.1	34	0.1	0.984068
V_{25}^{20}	50	0.1	50	0.1	0.997494
V_{29}^{29}	76	0.1	76	0.1	0.996217
V_{36}^{396}	542	0.6	542	0.3	0.997186
V_{44}^{528}	105	0.2	105	0.1	0.904577
V_{44}^{25}	87	0.2	87	0.1	0.974145
V_{64}^{78}	309	0.5	309	0.2	0.997506
V_{281}^{1300}	2491	4.2	2491	1.5	0.985928
V_{281}^{214}	2386	3.6	2386	0.6	0.987390
V_{80}^{1378}	****	****	54032	5.4	0.997120

1) Run on an Encore Multimax system.
‡ Link Reliability = 0.9.
**** Results not known.
Time is in CPU seconds.

TABLE V
COMPARISON OF EVALUATING RELIABILITY OF A NETWORK FROM PATHS AND CUTS USING CAREL (P II)[1]

Network	PATHSET			CUTSET		
	DPath	Time	$R(G)$†	DCut	Time	$Q(G)$††
V_4^4	4	0.0	0.978480	4	0.0	0.021520
V_9^2	7	0.0	0.968425	11	0.0	0.031575
V_9^5	11	0.0	0.997632	10	0.0	0.002368
V_{13}^9	16	0.0	0.977184	14	0.0	0.022816
V_{13}^{28}	15	0.0	0.964855	50	0.1	0.035145
V_{14}^{18}	23	0.1	0.996665	22	0.1	0.003335
V_{18}^{110}	94	0.2	0.994076	155	0.2	0.005924
V_{18}^{24}	30	0.1	0.969112	39	0.1	0.030888
V_{24}^{19}	39	0.1	0.975116	40	0.1	0.024884
V_{20}^{20}	34	0.1	0.984068	40	0.1	0.015932
V_{25}^{20}	50	0.1	0.997494	39	0.1	0.002506
V_{29}^{29}	76	0.1	0.996217	77	0.1	0.003783
V_{36}^{396}	542	0.3	0.997186	1319	1.2	0.002814
V_{44}^{528}	105	0.1	0.904577	3367	2.4	0.095423
V_{44}^{25}	87	0.1	0.974145	79	0.1	0.025855
V_{64}^{78}	309	0.2	0.997506	231	0.2	0.002494
V_{281}^{1300}	2491	1.5	0.985928	16524	13.6	0.014072
V_{281}^{214}	2386	0.6	0.987390	2339	0.5	0.012610
V_{80}^{1378}	54032	5.4	0.997120	317978	119.1	0.002880

1) Run on an Encore Multimax system.
† Link Reliability = 0.9.
†† Link unreliability = 0.1.
DPath (Dcut) refers disjoint path (disjoint cut).
Time is in CPU seconds.

obtains all possible E_i's for an F_i.

Lemma A.1: Assume two conditional cubes E_j and E_k of F_i. If $E_j \subseteq E_k$, then E_k is redundant. □

The RED operation implements Lemma A.1 and is performed for all (E_j, E_k) pairs. Note, the definition in Section III-B2 checks out $\bar{x}\bar{x} = \bar{x}$, and $\bar{x}(\bar{x}\bar{y}) = \bar{x}$ type redundancies. The nonredundant E_is form MCC. Besides removing redundancies, the RED operator partitions the MCC into IG's and DG's (refer to Section III-B3). Thus, the RED speeds up the computation time of the CAREL. The CMB operator for independent group (IG) uses $\bar{x} * \bar{y} = \bar{x}\bar{y}$ iteratively. As is obvious from Theorem A.1 (discussed later), $\bar{x} * \bar{y}$ is a special case of $X_i^{c_1} * X_j^{c_2}$. Here, we assume that no common elements are present with x and y. (x and y are independent.) The CMB operator for dependent group (DG) utilizes lemmas and theorems mentioned below.

Definition: An X_i represents a cube which could be an MCC E_i, or the one produced during CMB operation. For notational convenience, consider $X_i^c = X_i$ ($\overline{X_i}$), when c = 1 (0).

Lemma A.2: Assume $T_1^i, T_2^i, \cdots, T_k^i$ represent k partitions of X_i.

Then, $X_i^c = (T_1^i T_2^i \cdots T_k^i)^c$

$$X_i^c = \begin{cases} (T_1^i T_2^i \cdots T_k^i) & : c = 1 \\ \left(\overline{T_1^i}, T_1^i\overline{T_2^i}, \cdots, T_1^i T_2^i \cdots T_{k-1}^i \overline{T_k^i}\right) & : c = 0 \end{cases}$$

Proof: For $c = 1$, the result is obvious. With $c = 0$, DeMorgan's complementation law is rewritten so that the list of T_j^i's is collectively exhaustive. □

Theorem A.1: Consider I, L (J, K) as 2-partitions of X_i (X_j). We have $X_i^{c_1} * X_j^{c_2} = (IL)^{c_1}(JK)^{c_2}$. The terms $(IL)^{c_1}(JK)^{c_2}$ is

1) $X_i^{c_1} X_j^{c_2}$, when $X_i \cap X_j = \phi$, i.e., X_i and X_j are independent.

2) A, when $X_1 \cap X_2 = \phi$, i.e., $L = J$. Thus, $L = J$ represents a common term between X_1 and X_2.

For various combinations of c_1 and c_2, "A" is obtained as
i) *Case 1:* $[c_1 = c_2 = 2]$. $A = IJK$
ii) *Case 2:* $[c_1 = 1(0), c_2 = 0(1)]$.

$$A = \begin{cases} \phi & : I.K = \phi \\ I^{c_1} J K^{c_2} & : I.K \neq \phi \end{cases}$$

iii) *Case 3:* $[c_1 = c_2 = 0]$. $A = (\overline{J}, J\overline{I}\,\overline{K})$.

Proof: For $X_1 \cap X_2 = \phi$, $X_1^{c_1} * X_2^{c_2} = X_1^{c_1} X_2^{c_2}$ is straightforward. Use Lemma A.2 to show the results in Cases 1–3. With $c_1 = c_2 = 1$ and $L = J$, Case 1 is obvious. For $c_1 = 1, c_2 = 0$, $X_1^{c_1} * X_2^{c_2}$ is $(IJ)(\overline{JK})$ or $(IJ)(\overline{J}, J\overline{K})$. A result $(\overline{J}, J\overline{I})(JK)$ follows similarly when $c_1 = 0, c_2 = 1$. Thus, Case 2 of Theorem A.1 is proved. For $c_1 = c_2 = 0$, $X_1^{c_1} * X_2^{c_2} = (\overline{IJ})(\overline{JK})$. Using Lemma A.2, we interpret this result as $(\overline{J}, J\overline{I})(\overline{J}, J\overline{K})$ which after applying a Boolean identity [20] produces $(\overline{J}, J\overline{I}\,\overline{K})$.□

Lemma A.3: Note, $X * \phi = \phi * X = \phi$, where ϕ is a null set, and X represents any term.

Proof: Using Theorem A.1, the proof is obvious. □

Theorem A.2: For X_1, X_2, X_3, the CMB operator produces $X_1 \overline{X}_2 * \overline{X}_3 = GF(\overline{H}, H\overline{I}\,\overline{J})$ where G, F (I, H) represent 2-partitions for X_1 (X_2), and F, H, J are 3-partitions for X_3.

Proof: Note, $X_1\overline{X}_2$ is a term obtained considering $X_1^{c_1} * X_2^{c_2}$ in Theorem A.1 and represent mutually independent terms, i.e., X_1 and X_2 will have no term in common. Using Lemma A.1, $(X_1\overline{X}_2 * \overline{X}_3)$ is shown to be equal to $(G\,F\,\overline{IH}\,\overline{F}H\overline{J})$. Theorem A.2 is proved after applying Lemma A.2. □

From Theorems A.1 and A.2, it is clear that if we CMB k number of X_i's, we may generate a term of the type $X_1^{c_1} X_2^{c_2} \cdots X_k^{c_k}$. An iterative application of these theorems solves $(X_1^{c_1} X_4^{c_4} X_5^{c_5} \cdots X_k^{c_k}) * (X_2^{c_2} X_3^{c_3})$ as (*dots* $(X_1^{c_1} * X_2^{c_2}) * X_3^{c_3} * \ldots) * X_k^{c_k}$. Note, a CMB obtains *e.m.d.* events [we have called them as DTs].

Finally, GEN combines the F_i with the disjointing terms DT's. The operator utilizes five out of 12 possible combinations for $(-3^1, 0, 1)$ alphabets and is demonstrated to be complete (refer to the text). Hence, the algorithm CAREL is proved to be correct.

Acknowledgment

We thank the unknown referees for their helpful suggestions and comments which helped improve the readability of this paper.

References

[1] S. Hariri and C. S. Raghavendra, "SYREL: A symbolic reliability algorithm based on path cutset methods," *IEEE Trans. Comput.*, vol. C-36, pp. 1224–1232, Oct. 1987.
[2] S. Rai and D. P. Agrawal, *Distributed Computing Network Reliability*, IEEE Computer Society Press, 1990 (Tutorial Text).
[3] A. Grnarov, L. Kleinrock, and M. Gerla, "A new algorithm for network reliability computation," in *Proc. Computer Networking Symp.*, Dec. 1979, pp. 17–20.
[4] A. M. Johnson, Jr. and M. Malek, "Survey of software tools for evaluating reliability, availability and serviceability," *ACM Comput. Surveys*, pp. 227–269, Dec. 1988.
[5] D. Y. Koo, "D/Boolean application in reliability analysis," in *Proc. Annu. Reliability Maintainability Symp.*, 1990, pp. 294–301.
[6] V. Kini and D. P. Siewiorek, "Automatic generation of symbolic reliability functions for processor memory switch structure," *IEEE Trans. Comput.*, vol. C-31, pp. 752–771, Aug. 1982.
[7] C. C. Fong and J. A. Buzacott, "An algorithm of symbolic reliability computation with pathsets—or cutsets," *IEEE Trans. Reliability*, vol. R-36, pp. 34–37, Apr. 1987.
[8] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, MA: Addison-Wesley, 1974.
[9] C. J. Colbourn, *The Combinatorics of Network Reliability*, Oxford, England: Oxford University Press, 198?
[10] R. G. Bennetts, "Analysis of reliability block diagrams by Boolean techniques," *IEEE Trans. Reliability*, vol. R-31, pp. 159–166, June 1982.
[11] A. Rosenthal, "Approaches to comparing cut-set enumeration algorithms," *IEEE Trans. Reliability*, vol. R-28, pp. 62–65, Apr. 1979.
[12] S. Rai and K. K. Aggarwal, "An efficient method for reliability evaluation of a general network," *IEEE Trans. Reliability*, vol. R-27, pp. 206–211, Aug. 1978.
[13] J. A. Abraham, "An improved algorithm for network reliability," *IEEE Trans. Reliability*, vol. R-28, pp. 58–61, Apr. 1979.
[14] L. Fratta and U. G. Montanari, "A recursive method based on case analysis for computing network terminal reliability," *IEEE Trans. Commun.*, pp. 1166–1177, Aug. 1978.
[15] M. O. Locks, "A minimizing algorithm for the sum of disjoint products," *IEEE Trans. Reliability*, vol. R-36, pp. 445–453, Oct. 1987.
[16] X. Qian, N. Shirataori, and S. Noguchi, "An algorithm for evaluating K-terminal reliability," private communication.
[17] R. Sahner and K. Trivedi, "Performance and reliability analysis using directed acyclic graphs," *IEEE Trans. Software Eng.*, vol. SE-13, pp. 1105–1114, Oct. 1987.
[18] M. Veeraraghavan and K. S. Trivedi, "An improved algorithm for the symbolic reliability analysis of networks," in *Proc. Ninth Symp. Reliable Distributed Syst.*, Huntsville, AL, Oct. 1990, pp. 34–43.
[19] S. Rai and J. Tranan, "Computing network reliability of redundant MIN's," in *Proc. 21st Southeastern Symp. Syst. Theory*, Tallahassee, FL, Mar. 1989, pp. 221–225.
[20] R. Miller, *Switching Theory, Vol. I: Combinational Circuits*, New York: Wiley, 1965.
[21] V. K. Prasanna Kumar, S. Hariri, and C. S. Raghavendra, "Distributed program reliability analysis," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 42–50, Jan. 1986.
[22] R. K. Tiwari and M. Verma, "An algebraic technique for reliability evaluation," *IEEE Trans. Reliability*, vol. R-29, pp. 311–313, Oct. 1980.
[23] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, San Francisco, CA: Freeman, 1979.
[24] R. Johnson, "Network reliability and acyclic orientations," *Networks*, vol. 14, pp. 489–505, 1984.
[25] Y. B. Yoo and N. Deo, "A comparison of algorithms for terminal pair reliability," *IEEE Trans. Reliability*, vol. R-37, pp. 210–215, June 1988.
[26] J. S. Provan and M. Ball, "Computing network reliability in time polynomial in the number of cuts," *Oper. Res.*, vol. 32, pp. 516–526, 1984.
[27] J. S. Provan, "The complexity of reliability computation in planar and acyclic graphs," *SIAM J. Comput.*, vol. 15, pp. 694–702, 1986.
[28] S. Abraham and K. Padmanabhan, "Reliability of the hypercube," in *Proc. Int. Conf. Parallel Processing*, 1988, pp. 90–94.
[29] ——, "Performance of the direct binary n-cube network for multiprocessors," *IEEE Trans. Comput.*, vol. 38, pp. 1000–1011, July 1989.
[30] J. T. Blake and K. S. Trivedi, "Multistage interconnection network reliability," *IEEE Trans. Comput.*, vol. 38, pp. 1600–1604, Nov. 1989.

Sieteng Soh (S'90) received the B.S. degree in electrical engineering (computer option) in 19[illegible] from the University of Wisconsin, Madison, and the M.S. degree in electrical engineering in 1988 from Louisiana State University, Baton Rouge.

He is a Ph.D. student in the Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge. His research interests include computer architecture, fault-tolerant computing, parallel processing, and reliability of digital systems.

Mr. Soh is a student member of Eta Kappa Nu and Tau Beta Pi.

Suresh Rai (SM'86) received the B.E. degree from Benaras Hindu University in 1972, the M.E. degree from the University of Roorkee in 1974, and the Ph.D. from Kurukshetra University in 1980.

He joined the Regional Engineering College at Kurukshetra in 1974 and worked there for six years. After a brief stay at MMM Engineering College, Gorakhpur, he transferred to the University of Roorkee in 1981 as an Associate Professor. He also worked for two years at the School of Engineering at North Carolina State University, Raleigh. He is currently working with the Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge. He has taught and researched in the areas of reliability engineering, fault diagnosis, and parallel and distributed processing. He is a co-author of the book *Wave-shaping and Digital Circuits*, and tutorial texts *Distributed Computing Network Reliability* and *Advances in Distributed System Reliability*. He has guest edited a special issue of IEEE TRANSACTIONS ON RELIABILITY on the topic Reliability of Parallel and Distributed Computing Networks.

Dr. Rai has worked as a program committee member for IPCCC '91 (Phoenix) Conference. Currently, he is working as an Associate Editor for IEEE TRANSACTIONS ON RELIABILITY. He is a member of the Association for Computing Machinery.

IEEE TRANSACTIONS ON

RELIABILITY

1991 OCTOBER VOLUME 40 NUMBER 4 (ISSN 0018-9529)

A PUBLICATION OF THE IEEE RELIABILITY SOCIETY
and JOURNAL of the ASQC ELECTRONICS DIVISION

Special Issue — Design for Reliability of Telecommunication Systems & Services

A Computer Approach for Reliability Evaluation of Telecommunication Networks with Heterogeneous Link-Capacities

Suresh Rai, Senior Member IEEE
Louisiana State University, Baton Rouge
Sieteng Soh, Student Member IEEE
Louisiana State University, Baton Rouge

Key Words — Boolean technique, Clique, Cut set, Capacity related reliability, Path-graph, Path set, Telecommunication network, Terminal reliability

Reader Aids —
Purpose: Widen state of the art
Special math needed for explanations: Probability
Special math needed to use results: Same
Results useful to: Reliability analyst, network designer

Summary & Conclusions — The paper presents a computer approach to obtain a survivability index called capacity related reliability (CRR) in large telecommunication networks where links have different capacities. The proposed method is a 2-step approach. Step 1 deals with composite path enumeration (CPE). A k-composite path is defined as the union of the set of edges in any k simple paths and relates link capacity and network connectivity. The CPE approach, presented in the text, is an improvement over the algorithms of [1]. Step 2 manipulates k-composite paths information to generate the CRR. The paper uses CAREL [4] to solve this step. However, any existing techniques based on Boolean concept, probability theory, inclusion-exclusion principle, etc, [2-7] may also be utilized. The technique is automated using C on Encore Multimax System. The results on CRR for three networks with various values of minimum message capacity are presented in tables. An exhaustive technique is used to verify these results. However, an informal proof of the CPE approach is also included. Appendix A provides the implementation details of the technique. We have given counter examples (refer to Appendix B) for the algorithms of [10, 13] to show that these methods lead to incorrect conclusions under certain situations.

1. INTRODUCTION

To obtain the survivability index of a large telecommunication network, the network is modeled as a probabilistic graph $G(V,E)$, nodes (V) identify communicating centers, and links (E) represent connection services. Various measures for survivability index are presented in the literature [1-17]. They are characterized by different operational environments (OE) in which the network carries out its desired operation. As an example, in message switching or virtual circuit packet switching network, we need to establish a node to node connection in $G(V,E)$. As long as this 'connectedness' property of the network is maintained, one may successfully transmit messages (packets) through the network even if some of the nodes and/or links in $G(V,E)$ fail.

A network is generally validated for the connectedness of its OE by enumerating simple paths between all node pairs. In such situations, link capacity is often ignored or implicitly assumed to be equal and also large enough to sustain transmission of messages (packets) of any bandwidth (size). This assumption is unrealistic. The link capacity is a function of cost, and is limited. Each link in $G(V,E)$ may have different capacity. Moreover, there may be a minimum message capacity (W_{min}) requirement through the network. Obviously, the measure of connectedness using simple paths is not enough to validate this form of OE. This paper defines the concept of a k-composite path which captures the effect of message bandwidth, link capacity, and network connectivity. Note, a composite path satisfying an OE, where requisite amount of message bandwidth is also made available through the network having heterogenous link-capacities, is a success state of the network.

Recently, a few researchers have addressed the problem of capacity related reliability (CRR), or combining link capacity with terminal reliability. Doulliez & Jamoulle [8] have applied the decomposition principle to calculate the system reliability. They decompose the whole state space into three categories: a set of functioning states, a set of failed states, and a set of undetermined states. Each set of undetermined states is again decomposed into three categories, and so forth until the set of undetermined states is null. This implies keeping track of numerous sets of undetermined states as well as the relevant upper and lower limiting states of each set. Hence it requires large memory sizes for large systems. Lee's technique [9] uses a labelling scheme to route the flow through the network. It is, however, suited to acyclic graphs only. Its adaptation to mixed and cyclic graph imposes a problem because of the existence of feedback [an unsuitable situation for the labelling scheme]. Misra & Prasad [10] utilize a failure path list to enumerate a term like composite path defined in the text. The success of a 2-composite path is tested against a given W_{min}. Next, the method [10] proceeds with the failure paths which have not produced any success 2-composite paths, and combines them to generate higher order composite paths. Each iteration checks for success using W_{min} and removes the simple paths which have already combined to produce success composite paths. The method [10] terminates when no more simple path is available to generate k-composite paths. A counter example, presented in Appendix B, proves that [10] fails in general to give correct result. Moreover, [10] does not provide

This work has been partially supported by the US Air Force Office of Scientific Research under grant AFOSR-91-0025. A Preliminary version of this paper was presented at the 1991 Annual Reliability and Maintainability Symposium [1].

0018-9529/91/1000-0441$01.00©1991 IEEE

any procedure to compute the capacity of a composite path which, in turn, is useful to decide the success or failure state of the network. Recently, [12] proposes a technique to generate the CRR. Aggarwal [13] mentions that the method in [12] lacks generality and provides incorrect results. We have given a counter example (refer to Appendix B) to show that the algorithm in [13] leads to a wrong conclusion too. The basic problem with [12,13] lies in the procedure used to compute composite path capacity. Le & Li [20] have addressed the problem of reliability of networks with dependent failures and multimode components. They assign link capacities and obtain a numerical figure, not the reliability expression, for the network. Rai, et al [11] and Rueger [14] have proposed algorithms which obtain a symbolic CRR expression under a capacity constraint. However, their algorithms suffer from the drawback of generating a huge number of redundant paths/cuts. They are, thus, impractical even for moderate sized graphs where the number of paths is more than ten. Recently, Rai & Soh [1] have proposed two algorithms to enumerate composite paths. The composite path enumeration technique, presented in this paper, is an improvement over the algorithms of [1]. Section 5 provides a detailed discussion.

The layout of the paper is as follows: Section 2 describes the background material. Section 3 presents composite path concepts and the issues related to its enumeration. It also introduces several definitions and theorems which are useful in reducing the complexity of the composite path enumeration (CPE) approach. Section 4 describes the CPE technique that is further illustrated by an example. An informal proof of the algorithm and its time complexity analysis are also given in this section. Section 5 discusses the experimental results on the CRR index, obtained using CAREL[4], for three networks with various values of W_{min}. Finally, Appendix A provides an implementation technique for the CPE algorithm using bit vector representation and Appendix B gives the counter examples to the methods [10,13].

2. PRELIMINARIES

In the graph model $G(V,E)$ of a telecommunication network, consider an edge j has a finite capacity w_j which is known a priori. Let l be the total number of edges in $G(V,E)$. A flow in a network is a function assigning a non-negative number f_j to each edge j so that $f_j \le w_j$, and for a vertex (that is neither source nor terminal) the in- and out-flow are the same (flow conservation). Note, w_j provides a bound on flow passing through edge j. The network is good if and only if a specified amount of signal capacity (W_{min}) can be transmitted from the input to the output node or (s,t) node pair.

An edge j is said to be UP (DOWN) if it is functioning (failed). An UP (DOWN) edge is denoted by j ($\bar{j}$). An (s,t) cut is a disconnecting set. All communication between a prescribed (s,t) node pair is disrupted once the edges in (s,t) cut fail. An (s,t) cut i, C_i, is minimal if no proper subset of it represents a 'cut'. The cut set $C_{s,t}$ is the set of all minimal cuts for the graph $G(V,E)$. Let the total number of cuts be n. The capacity of a cut set, $W(C_i)$, for a minimal cut C_i is the sum of capacities of edges in C_i. From max-flow min-cut theorem [18], the maximum capacity flow, W_{max}, through the graph $G(V,E)$ is:

$$W_{max} = \min_i \{W(C_i)\}. \tag{1}$$

As an example, consider a bridge network shown in figure 1. The cut set $C_{s,t}$ is $\{(1,2), (1,3,5), (2,3,4), (4,5)\}$. Using [18], the capacities are $W(C_1) = 14$, $W(C_2) = 19$, $W(C_3) = 12$, and $W(C_4) = 7$. Here, edge capacities are $w_1 = 10$, $w_2 = 4$, $w_3 = 5$, $w_4 = 3$, and $w_5 = 4$. Now, using (1), $W_{max} = \min\{14, 19, 12, 7\} = 7$.

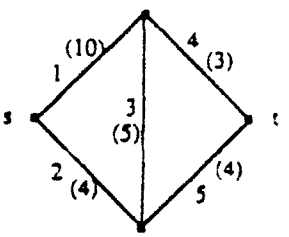


Figure 1. Bridge Network
[The link capacity is shown within ( ).]

A simple path i, P_i, for an (s,t) node pair is formed by the set of UP edges such that no node is traversed more than once. Note, any proper subset of simple paths does not result in a path between these two node pairs. The path set $P_{s,t}$ is a set whose elements are simple paths. Let m be the total number of simple paths in $G(V,E)$. The capacity of a simple path P_i, $W(P_i)$, is obtained from the capacities of UP edges contained in P_i and is [11]:

$$W(P_i) = \min_{j\,:\,j \in \{P_i\}} \{w_j\}. \tag{2}$$

The capacity for path $P_1 = (1,4)$ in figure 1 is $W(P_1) = \min\{w_1, w_4\} = 3$.

If $W(P_i) \ge W_{min}$, the P_i, in addition to satisfying the connectivity requirements, fulfills the capacity constraint too. The path P_i is, then, called a success state of $G(V,E)$. Otherwise, the P_i represents a failure state. Note, in the event that edge capacities are infinitely large, all simple paths form success states because they do provide (s,t) connectivity, and their successes ensure the network success. However, for a finite capacity situation, all simple paths may or may not lead to the success states of $G(V,E)$. Depending on W_{min}, some or all simple paths may fail to satisfy the capacity constraint. Thus, simple path (minimal cut), an important concept in terminal reliability, has to be revisited while considering the CRR measure. The concept of composite path (introduced in Section 3) is a step in this direction.

3. COMPOSITE PATH

3.1 *Concept and Issues*

Definition: A k-composite path $CP_I(k)$ is defined as the union of set of edges in any k simple paths P_i's, where $i \in I$, and $(1 \le k \le m)$. □

Note, a k-composite path describes a subgraph of $G(V,E)$. Moreover, all simple paths represent k-composite paths where $k=1$.

Example. Consider the bridge network in figure 1. The path set $P_{s,t}$ of the network is : P_1 : (1,4), P_2 : (1,3,5), P_3 : (2,5), and P_4 : (2,3,4). From the $P_{s,t}$ we generate the 2-composite paths:

$$CP_{1,2}(2) = (1,3,4,5),\ CP_{1,3}(2) = (1,2,4,5),$$

$$CP_{1,4}(2) = (1,2,3,4),\ CP_{2,3}(2) = (1,2,3,5),$$

$$CP_{2,4}(2) = (1,2,3,4,5),\ \text{and}\ CP_{3,4}(2) = (2,3,4,5).$$

Lemma 1. For m simple paths representing (s,t) connectedness of the network, the total number of possible k-composite paths is (2^m-1). □

Misra & Prasad's technique [10] reduces the generation of all (2^m-1) possible k-composite paths. However, a counter example in Appendix B shows the method in [10] lacks generality. Another problem in composite-path enumeration stems from the capacity computation for the $CP_I(k)$. The capacity of a simple path $W(P_i)$ is obtained easily from (2). However, we need to devise an efficient technique to get the capacity of a $CP_I(k)$ for $k > 1$. Ref [10] does not discuss any method. Papers [12,13] have proposed techniques to help obtain the capacity of a $CP_I(k)$. However, [13] mentions that the method in [12] is not correct. Appendix B provides a counter example to show that [13] also fails to generate correct results under certain situations. We provide a lemma to evaluate the capacity of a composite path $CP_I(k)$, for $k > 1$. For this, we introduce the concept of Composite Path Cut (CPC).

Definition: A $CPC_I(j)$ is a modified (s,t) cut C_j for the graph $G(V,E)$ and is defined for a composite path $CP_I(k)$. The $CPC_I(j)$ is:

$$CPC_I(j) = CP_I(k) \cap C_j\ ;\ j=1,\ldots,n. \tag{3}$$

□

Since a $CP_I(k)$ describes a subgraph of $G(V,E)$, the $CPC_I(j)$ represents a cut for the $CP_I(k)$ induced graph. The failure of the edges in the $CPC_I(j)$ leads to $CP_I(k)$ communication disruption between a prescribed (s,t) node pair. Note, there exists n number of $CPC_I(j)$'s for a composite path $CP_I(k)$.

Lemma 2. The weight of a composite path, $W(CP_I(k))$, is:

$$W(CP_I(k)) = \min_{j=1,\ldots,n} \{W(CPC_I(j))\} \tag{4}$$

where $W(CPC_I(j))$ represents the weight of a $CPC_I(j)$ and is obtained by applying (1) to various $CPC_I(j)$'s. Note, for $k=1$, Lemma 2 gives the same results as obtained by (2). □

Example. Consider figure 4 and table 1 where a composite path $CP_I(2) = (1,2,5,6)$ has $CPC_I(j)$s as {(1,2), (5,6), (1,5), (1,5), (2,6), (5,6), (2,6)}. The weight of the $CP_I(2)$ is, thus, 11 units.

TABLE 1
Pathset and Cutset for Figure 4

Pathset	Cutset
$P_1 = (1,6)$	$C_1 = (1,2)$
$P_2 = (2,5)$	$C_2 = (5,6,8)$
$P_3 = (1,7,8)$	$C_3 = (1,3,5,8)$
$P_4 = (1,3,5)$	$C_4 = (1,3,4,5)$
$P_5 = (2,4,8)$	$C_5 = (2,3,6,7)$
$P_6 = (2,3,6)$	
$P_7 = (1,4,5,7)$	$C_6 = (4,5,6,7)$
$P_8 = (2,3,7,8)$	$C_7 = (2,3,4,6,8)$
$P_9 = (1,3,4,8)$	

Definition: A composite path $CP_I(k)$ is a success state of the network if it satisfies the capacity or flow constraint:

$$W(CP_I(k)) \ge W_{min}. \tag{5}$$

Otherwise, the $CP_I(k)$ is a failure state. Moreover, a $CP_I(k)$ is defined as a redundant state of the network if there is at least one success state $CP_I(u)$ such that $CP_I(u) \subseteq CP_I(k)$. □

Definition: A *cross_link*$_I(k)$, defined for the composite path, is the set of links common to the k simple paths forming the $CP_I(k)$. □

Definition: The weight of a *cross_link*$_I(k)$, W(*cross_link*$_I(k)$), is computed following (2). □

The notion of a *cross_link* and its weight are used to detect a failure k-composite path a priori.

Theorem 1. $CP_I(k)$ is a failure state if W(*cross_link*$_I(k)$) $< W_{min}$, for *cross_link*$_I(k) \ne \Phi$. □

Proof. Since every element in *cross_link*$_I(k)$ is a link in all the k paths that form the $CP_I(k)$, the flow in $CP_I(k)$ is limited to the weight of the *cross_link*$_I(k)$. *Q.E.D.*

Thus, before generating the $CPC_I(j)$ set for a $CP_I(k)$, use Theorem 1 to see if the $CP_I(k)$ is a failure state. Obtain the $CPC_I(j)$s only when the *cross_link* concept fails to detect a failure state. This happens when W(*cross_link*$_I(k)$) $\ge W_{min}$ or the k simple paths have no links in common, ie, *cross_link*$_I(k) = \Phi$. Initially, each *cross_link*$_I(1)$ of $CP_I(1)$ is given by simple paths in *F_set*(1). Later, for each $CP_I(k)$, update the *cross_link*$_I(k)$ by taking the set theoretic intersection of two *cross_link*'s of $CP_I(k-1)$s that merge into $CP_I(k)$s.

3.2 *Development of the Technique*

The proposed CPE algorithm starts from the failure paths and utilizes divide and conquer strategy to reduce the complexity of the problem. In Section 2, we discussed success and failure states of a network from the aspect of simple paths. For a given W_{min} capacity, first, test all simple paths and partition them into functioning (*S_set*) and non-functioning (*F_set* (1)) groups. Let the number of elements in *F_set*(1) be β. The *F_set*(1) is then used to obtain k-composite paths, $1 < k \leq \beta$. Lemma 2 (and the definition following the lemma) is utilized to partition $CP_i(k)$s into the *S_set* and *F_set*(k) categories. Note, both the *S_set* and *F_set*(k) can have many redundant terms. The redundancy (duplication and absorption) is checked easily with the help of a Boolean identity $A \cup AB = A$. However, if special care is taken in the enumeration procedure, many of the redundancies can be avoided in the first place. We have observed that the number of redundant composite paths is reduced when the concept of path composition is recursively applied only to the failure states of the network. Additionally, use a path_graph PG(V,E) to obtain higher order ($k > 2$)compositions.

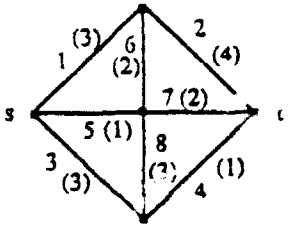


Figure 2. 5-Node, 8-Link Network
[The link capacity is shown within ( ).]

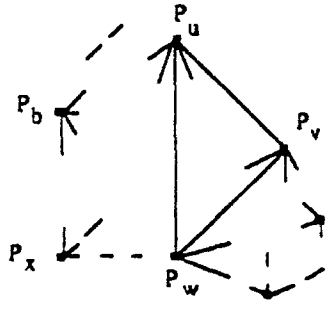


Figure 3. Path-Graph PG(V,E)

Figure 3 illustrates a path_graph PG(V,E). Here, a node P_u belongs to *F_set*(1). An edge joining any two nodes P_u with P_v is defined by a 2-composite path in *F_set*(2). Note that P_u $P_vP_w(P_uP_vP_wP_x)$ generates a 3(4)-composition. It is reflected by a 3(4)-node complete graph or clique [19] in PG(V,E). In general, a j-composite path is a j-clique in the path_graph. The problem of composite path enumeration is, thus, similar to determining cliques in PG(V,E). Note, each state in *F_set*(j) is a clique. Thus, generating higher order cliques is straightforward. Two $(j-1)$-cliques may form a j-clique. Take XOR set theoretic operation on the elements of the two lower order cliques. If the result is an edge in PG(V,E), the two $(j-1)$-cliques produce a j-clique. The elements of the j-clique are obtained by using union set theoretic operation on the elements of the $(j-1)$-cliques. This process is further explained in Section 4.

3.3 *Efficient Generation of 2-Composite Paths*

Algorithm 1 in [1] generates $\beta(\beta-1)/2$ 2-composite paths or links in PG(V,E). To further reduce the complexity of this algorithm, we use the concepts of *key_cut*, *key_links*, and *path_groups* defined as follows.

Definition: A minimal cut C_i is defined as *key_cut* if: i) $w_j < W_{min}$, for all $j \in C_i$, and ii) $(w_j + w_l) \geq W_{min}$ for at least one j, l pair; $j, l \in C_i$. □

Condition i is required, while condition ii [$(w_j + w_l) \geq W_{min}$] is important as it helps identify the failure composite paths a priori. If condition ii fails ($(w_j + w_l) < W_{min}$ for all j, l pair) the 2-compositions become failure states for the network. All failing simple paths are, then, used to generate $\beta(\beta-1)/2$ failure 2-composite paths as discussed in subsection 3.2. To illustrate a *key_cut*, consider a minimal cut (4,5) for the bridge network in figure 1. Capacities of links 4 and 5 are individually less than W_{min} ($=6$) while the capacity of the minimal cut (4,5) is greater than W_{min}. Thus, the cut (4,5) represents a *key_cut*.

Definition: The cardinality of a *key_cut* (denoted as |*key_cut*|) is the total number of elements in it. Select C_i as a *key_cut* if —

i. C_i is either a source-node or terminal-node cut. If not, C_i should have at least one link connected to the source or the terminal node, and

ii. $|C_i|$ is minimum. If $|C_i|$ is equal for two or more terms, take arbitrarily one C_i. □

Definition: The elements of a selected *key_cut* are termed as *key_links*. □

Definition: A *path_group* $G(\eta)$ is the set of failure paths, each of which contains a *key_link* η. □

Without loss of generality, a failure path which contains two or more *key_links*, $\{\alpha,\ldots,\eta,\ldots,\gamma\}$ is assigned to a *path_group* $G(\eta)$, where η is lexicographically smallest of $\{\alpha,\ldots,\eta,\ldots,\gamma\}$. This allocation strategy is useful to minimize the number of *path_groups* generated.

Example. Consider all four simple paths (1,4), (2,5), (1,3,5), (2,3,4) in figure 1 as failure states. Grouping them with *key_links* 4 and 5 we have:

$$G(4) = \{(1,4), (2,3,4)\} \text{ and } G(5) = \{(2,5), (1,3,5)\}.$$

Theorem 2. A k-composite path $CP_i(k)$ within a *path_group* $G(\eta)$ is a failure state. □

Proof. From the definition of a *path_group* $G(\eta)$, a *cross_link*$_i(k)$ obtained from the $CP_i(k)$ contains at least one

key_link. Using Theorem 1 and the definition of a *key_cut*, the $W(CP_I(k))$ is less than W_{min}. *Q.E.D.*

Theorem 3. The upper bound on the number of 2-composite paths is:

$$\sum_{i,j}^{\gamma} \beta_i \beta_j \; ; \; i \neq j, \; i < j.$$

Here β_i and β_j are $|G(i)|$ and $|G(j)|$, respectively. Moreover, $\sum_i \beta_i = m$ (*or* β) and γ is the total number of groups. □

Proof. The number of 2-composite paths N_2 is

$$\leq \frac{m^2}{2} - \sum_{i=1}^{\beta} \beta_i^2/2$$

which, if solved, gives the result. *Q.E.D.*

Theorem 4. The number of k-composite paths N_k is bound by $N_k/N_{k-1} \leq m-k+1/k$, where N_{k-1} represents the number of $(k-1)$-composite paths. □

Proof. For the upper bound, Theorem 4 is true considering N_k and N_{k-1} as binomial coefficients $\binom{m}{k}$ and $\binom{m}{k-1}$. Theorem 3 shows that N_2 is less than $\binom{m}{2}$. Thus, $N_3/N_2 < m-2/3$. *Q.E.D.*

Note, to generate k-composite paths ($k>2$), we also have to consider all $(k-1)$-composite paths formed from the elements within a *path_group*. We call the set of such composite paths an *F_set_dummy*$(k-1)$. Although all states in *F_set_dummy*$(k-1)$ are failure states (refer to Theorem 2), they can generate success states to obtain higher order states (cliques).

4. COMPOSITE-PATH ENUMERATION TECHNIQUE

4.1 *CPE Algorithm*

Utilizing concepts in subsections 3.1 through 3.3 a CPE algorithm is developed as follows.

1. Read input files having
   - path set and cut set;
   - link capacities w_j, for $j=1,\ldots,l$;
   - W_{min};
2. Get $W_{max} = \min\{W(C_i)\}$;
   if ($W_{max} < W_{min}$) then goto 5;

3a. Determine success/failure states from path set; create *F_set*(1) and *S_set* for failure and success paths, respectively;

3b. If ($|F\_set(1)| > 1$) then
   find a *key_cut*;
   else goto 5;

3c. If (*key_cut* exists) then
   begin
   create *path_groups*;
   generate $CP_I(2)$'s using the *path_groups*;
   create *F_set_dummy*(2) formed by 2-composite paths within each *path_group*;
   end;
   else
   begin
   generate $CP_I(2)$'s directly from *F_set*(1);
   F_set_dummy(2) = { }
   end;

3d. Determine success/failure states for all $CP_I(2)$s; /* use (4) */
   append non-redundant success, $CP_I(2)$s to *S_set*;
   append non-redundant failure, $CP_I(2)$s to *F_set*(2);

4. Initialize PG_set = *F_set*(2) ∪ *F_set_dummy*(2);
   for ((k=3...β) and (|*F_set*(k−1)| > 1)) do
   begin
   /* Use PG_set to test for a k-composite path following the method described in subsection 3.2 */
   generate non-redundant $CP_I(k)$'s from *F_set*$(k-1)$;
   generate non-redundant $CP_I(k)$'s from combination of *F_set*$(k-1)$ with *F_set_dummy*$(k-1)$;
   /* for each $CP_I(k)$ */
   if ($W(cross\_link_I(k)) < W_{min}$) then
   success = FALSE;
   else
   begin
   obtain $CPC_I(j)$ set for each $CP_I(k)$;
   if ($\min_{j=1,\ldots,n} \{W(CPC_I(j))\} \geq W_{min}$) then
   success = TRUE; /* $CP_I(k)$ is a success */
   end
   if (success) then
   append the success $CP_I(k)$ to *S_set*;
   else
   append the failure $CP_I(k)$ to *F_set*(k);
   create *F_set_dummy*(k) from *F_set_dummy*$(k-1)$;
   F_set(k) = *F_set*(k) ∪ *F_set_dummy*(k);
   end;

5. end.

4.2 *Proof of Correctness*

The CPE algorithm solves two main problems:

1. Generating sufficient number of $CP_I(k)$'s which should lead to all possible success composite paths in a network.

2. Evaluating the capacity of a composite path to check for success/failure states.

In what follows, we show that the CPE algorithm always gets a solution, if it exists.

- Correctness proof for solving problem 1.

Lemma 3. Let $CP_I(k)$ and $CP_J(c)$ be success states. If $CP_I(k) \subseteq CP_J(c)$ then $CP_J(c)$ is a redundant success state and, hence, need not be generated. □

The CPE algorithm starts from failure simple paths. Following Lemma 3, we need not generate higher order of composite paths from success simple paths. The CPE algorithm generates sufficient $CP_I(2)$'s to obtain all possible success $CP_I(2)$'s (refer to subsection 3.3). For generating higher order states,

the CPE utilizes path-graph PG(V,E) whose nodes and links are *F_set*(1) and *F_set*(2), respectively. The problem of enumerating 3-, 4-, ..., k-composite paths is conceptually equivalent to generating 3-, 4-, ..., k-cliques in the path-group PG(V,E). Following Lemma 3, it is obvious that if *key_cut* does not exist, the CPE algorithm generates sufficient k-cliques since it always generate k-cliques from all combinations of failure $(k-1)$-cliques. When a *key_cut* exists, the CPE uses *key_cut*, *key_links*, and *path_groups* to recognize some failure composite paths a priori. Since the CPE maintains *F_set_dummy*(k) to keep those 'predicted' failure k-composite paths and later utilizes it to form $(k+1)$-cliques with *F_set*(k), for this case, the CPE also produces sufficient k-cliques.

- Correctness proof for solving problem 2.

The algorithm uses (2) (a special case of Lemma 2) and Lemma 2 to evaluate the capacity of simple paths and $CP_l(k)$'s, respectively. Note a k-composite path is a subgraph of the network $G(V,E)$. Following the definition of $CPC_l(j)$ (refer to subsection 3.1), it is obvious that the $CPC_l(j)$ set represents the cutsets for the $CP_l(k)$ induced graph. Thus, by using max-flow min-cut theorem [18] on $CPC_l(j)$, we obtain the max-flow or capacity of the subgraph $CP_l(k)$.

4.3 *Computational Time Complexity*

It is shown in [3] that network reliability is an NP-complete problem. The CRR problem is a network reliability problem and, hence, it is difficult to compute its complexity. The CPE technique is an algorithm whose performance depends on the number of $CP_l(k)$'s generated, which in turn relies on the topology, link capacities, and minimum message bandwidth (W_{min}) for the given network. Note, Appendix A provides the subroutines that are used to implement the CPE technique, and shows that each subroutine is a polynomial time function. In the following, we give the complexity of the CPE algorithm using some notation defined below. For a given network and a certain state k, let g_k and h_k be the number of non-redundant success and failure $CP_l(k)$'s, respectively. Let r be the total number of redundant $CP_l(k)$'s enumerated in the network, and K be the highest k-composition generated ie, $1 \le k \le K$. Note, in general, K is less than the total number of paths m. The total number of states generated for a network is, thus, computed as:

$$r + \sum_{k=2}^{K} h_k + g_k. \text{ Let } g = \sum_{k=2}^{K} g_k \text{ and } h = \sum_{k=2}^{K} h_k.$$

It is obvious that $g \le m$.

For each step k, each $CP_l(k)$ is first checked for its redundancy against all success composite-paths and $CP_l(k)$s generated so far. Note, using set theoretic operation, a $CP_l(k)$ can be tested for its redundancy against any $CP_l(k)$ in constant time. Thus, the time complexity to detect the r number of redundant $CP_l(k)$s is:

$$t_r = O\left(mr + \sum_{k=2}^{K} r_k h_k\right),$$

where r_k denotes the number of redundant states in step k. The capacity of a $CP_l(k)$ is computed in $O(n)$ (refer to Appendix A) and, thus, the capacity computation for all non-redundant states enumerated is given as $t_c = O(mn + hn)$. Given t_r and t_c, the time complexity of the CPE algorithm is:

$$t_{CPE} = t_r + t_c = O\left(mn + mr + \sum_{k=2}^{K} r_k h_k + hn\right).$$

It is obvious that the bottleneck of CPE technique lies in the number of failure k-composite paths h which can reach up to $O(2^m)$.

4.4 *Illustrating Example*

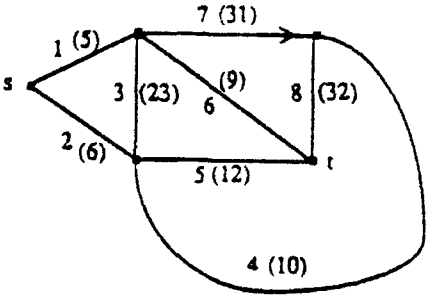


Figure 4. 5-Node, 8-Link Network
[The link capacity is shown within ( ).]

For figure 4, let the link capacities be:

1	2	3	4	5	6	7	8	links
5	6	23	10	12	9	31	32	link capacity

Various assumptions are given in [1]. It is also assumed that (s,t) path set and cut set be given. Table 1 illustrates these sets of information. Using (1), we get $W_{max} = 11$. The weight of each path is checked against a given value of W_{min} ($=10$). It is found that none of the eight paths forms the success state for the network/system. One way to form 2-compositions, $CP_l(2)$'s, is to generate all $\binom{9}{2} = 36$ composite paths of size 2. Alternatively, we may define a *key_cut* from the cut set. In the present example, the cut $C_1 = (1,2)$ qualifies to be the *key_cut*. Consider its elements 1 and 2 as the *key_links*, and partition the 9 failure paths into 2 groups as:

$$G(1) = \{(1,6), (1,7,8), (1,3,5), (1,4,5,7), (1,3,4,8)\};$$

$$G(2) = \{(2,5), (2,4,8), (2,3,6), (2,3,7,8)\}.$$

Here $|G(1)|$, and $|G(2)|$ are 5 and 4, respectively. Thus, we need to generate only (36-16) = 20 2-composite paths. They are:

$CP_{1,2}(2)$, $CP_{1,5}(2)$, $CP_{1,6}(2)$, $CP_{1,8}(2)$, $CP_{2,3}(2)$, $CP_{2,4}(2)$, $CP_{2,7}(2)$, $CP_{2,9}(2)$, $CP_{3,5}(2)$, $CP_{3,6}(2)$, $CP_{3,8}(2)$, $CP_{4,5}(2)$, $CP_{4,6}(2)$, $CP_{4,8}(2)$, $CP_{5,7}(2)$, $CP_{5,9}(2)$, $CP_{6,7}(2)$, $CP_{6,9}(2)$, $CP_{7,8}(2)$, $CP_{8,9}(2)$.

Here, all $CP_i(2)$'s but for $CP_{4,8}(2)$ are success states for the network and, hence, we obtain $F\_set(2) = \{CP_{4,8}(2)\}$. Since $|F\_set(2)| = 1$, we need not generate 3- or higher order composite paths. Substituting for various $CP_i(2)$'s from table 1, and deleting the redundancies, we obtain 8 composite paths:

{(1,2,5,6), (1,2,4,6,8), (1,2,5,7,8), (1,2,3,5), (1,2,4,5,7), (1,2,4,7,8), (1,2,3,4,8), and (1,2,3,7,8)}.

Using CAREL [4], the CRR expression is:

$$CRR = p_1p_2p_3p_5 + p_1p_2p_5p_6(1-p_3) + p_1p_2p_3p_7p_8(1-p_5) + p_1p_2p_4p_7p_8(1-p_3)(1-p_5p_6) + p_1p_2p_4p_5p_7(1-p_3)(1-p_6)(1-p_8) + p_1p_2p_5p_7p_8(1-p_3)(1-p_4)(1-p_6) + p_1p_2p_4p_5p_8(1-p_5)(1-p_7) + p_1p_2p_3p_4p_8(1-p_5)(1-p_6)(1-p_7).$$

Substituting known values for link reliability p_i's, we obtain a numerical value for CRR. For example, if $p_i = 0.9$ for all i, CRR = 0.799655.

5. DISCUSSION

The proposed CPE technique is simple, and is implemented in C on an Encore Multimax system. Tables 2a, 3a, and 4a illustrate the number of failure simple paths, success states, and reliability values (CRRs) for the nerworks shown in figures 4–6, respectively. Various W_{min} values are considered to generate these sets of information. To check for the correctness of the result, we have compared the success states obtained using our CPE algorithm with those obtained by [1, algorithms 1 & 2] and an exhaustive method. Note, we refer to [1, algorithms 1 & 2] as Alg1 and Alg2 respectively. For the exhaustive method, we generate all possible combinations of failure simple paths (which equal to $2^m - m - 1$ states in the worst case). Lemma 2 is used to check each of the states to determine success states. Delete the redundant terms, if any. The results generated from our CPE algorithm exactly match with those obtained from Alg1, Alg2, and the exhaustive method. On Encore Multimax system, the CPU time for different entries in table 4a ranges from 0.1 to 0.3 seconds, while it took 27.6 seconds CPU time (more than 4 hours real time) using exhaustive method (generating $2^{25} - 26$ states). Note, for all j, $\min\{w_j\} < W_{min} \leq W_{max}$ since for W_{min} smaller or equal to $\min\{w_j\}$, all the simple paths will always be success states.

TABLE 2a
Results for Figure 4

No.	W_{min}	Failure Paths	Success	CRR #
1	6	5	4	0.897407
2	7	9	9	0.806806
3	8	9	9	0.806806
4	9	9	9	0.806806
5	10	9	8	0.799655
6	11	9	7	0.799064

TABLE 2b
Results for Figure 4 (Continued)

		$CP_i(k)$s Generated			
No.	W_{min}	Exhaustive	Alg1	Alg2.CPE	'Saving' of CPE*
1	6	26	36	36	7
2	7	502	73	20	0
3	8	502	73	20	0
4	9	502	73	20	0
5	10	502	73	20	1
6	11	502	73	21	3

#link reliability = 0.9 for all cases.
*The difference between $CPC_i(k)$s generated by Alg2[1] and CPE

TABLE 3a
Results for Figure 5

No.	W_{min}	Failure Paths	Success	CRR #
1	2	5	13	0.977923
2	3	10	16	0.974977
3	4	13	29	0.949844
4	5	14	26	0.857353
5	6	14	15	0.815038
6	7	14	13	0.737874
7	8	14	8	0.586283
8	9	14	4	0.449795
9	10	14	1	0.313811

TABLE 3b
Results for Figure 5 (Continued)

		$CP_i(k)$s Generated			
No.	W_{min}	Exhaustive	Alg1	Alg2.CPE	'Saving' of CPE*
1	2	26	21	21	5
2	3	1013	88	88	7
3	4	8178	227	227	17
4	5	16369	733	733	52
5	6	16369	1166	751	55
6	7	16369	1501	1043	55
7	8	16369	1934	1436	55
8	9	16369	2139	949	125
9	10	16369	2160	1565	125

#link reliability = 0.9 for all cases.
*The difference between $CPC_i(k)$s generated by Alg2[1] and CPE

TABLE 3a
Results for Figure 5

No.	W_{min}	Failure Paths	Success	CRR #
1	2	5	13	0.977923
2	3	10	16	0.974977
3	4	13	29	0.949844
4	5	14	26	0.857353
5	6	14	15	0.815038
6	7	14	13	0.737874
7	8	14	8	0.586283
8	9	14	4	0.449795
9	10	14	1	0.313811

TABLE 3b
Results for Figure 5 (Continued)

		$CP_i(k)$s Generated			
No.	W_{min}	Exhaustive	Alg1	Alg2,CPE	'Saving' of CPE*
1	2	26	21	21	5
2	3	1013	88	88	7
3	4	8178	227	227	17
4	5	16369	733	733	52
5	6	16369	1166	751	55
6	7	16369	1501	1043	55
7	8	16369	1934	1436	55
8	9	16369	2139	949	125
9	10	16369	2160	1565	125

#link reliability = 0.9 for all cases.
*The difference between $CPC_i(k)$s generated by Alg2[1] and CPE

TABLE 4a
Results for Figure 6

No.	W_{min}	Failure Paths	Success	CRR #
1	7	24	45	0.973055
2	8	25	41	0.962361
3	9	25	41	0.962361
4	10	25	41	0.962361
5	11	25	27	0.934972
6	12	25	32	0.934972
7	13	25	15	0.845304
8	14	25	15	0.691823
9	15	25	15	0.691823
10	16	25	15	0.691823
11	17	25	11	0.648671
12	18	25	10	0.640535
13	19	25	11	0.626502

TABLE 4b
Results for Figure 6 (Continued)

		$CP_i(k)$s Generated			
No.	W_{min}	Exhaustive	Alg1	Alg2,CPE	'Saving' of CPE*
1	7	2^{24}-25	724	724	90
2	8	2^{25}-26	1055	341	135
3	9	2^{25}-26	1055	341	135
4	10	2^{25}-26	1055	341	135
5	11	2^{25}-26	1766	493	180
6	12	2^{25}-26	1766	493	180
7	13	2^{25}-26	3831	1308	180
8	14	2^{25}-26	5568	2124	179
9	15	2^{25}-26	5568	2124	179
10	16	2^{25}-26	5568	2124	179
11	17	2^{25}-26	5653	2190	179
12	18	2^{25}-26	5656	2195	179
13	19	2^{25}-26	5671	2209	179

#link reliability = 0.9 for all cases.
*The difference between $CPC_i(k)$s generated by Alg2[1] and CPE

Tables 2b, 3b, and 4b show the number of composite paths generated to produce the non-redundant success paths for the networks in figures 4–6, respectively. Note, the total number of composite paths generated depends on the given value W_{min} and link capacities of the network as well as on the network topology and, hence, is difficult to determine. The tables compare the performances of our CPE algorithm with Alg1, Alg2, and the exhaustive method in term of the total number of composite paths generated to obtain those success states of the networks. As anticipated, our CPE algorithm outperforms both exhaustive and Alg1 [1]. Notice that for a given value of W_{min}, the number of states generated by the CPE algorithm and both Alg1 and Alg2 are the same when *key_cut* does not exist in the network. Except for table 4b, the exhaustive method enumerates all possible composite paths. To speed up our experiments, for figure 6, the exhaustive method generates only up to all possible combinations of 6 simple paths (the network has at most 4-composite non-redundant success paths obtained by our CPE algorithm and both Alg1 and Alg2).

The last column of tables 2b, 3b, and 4b display the 'savings' of CPE method over Alg2 for various W_{min}. Since the total number of states generated in both methods is the same, we compare their performances based on the number of $CPC_i(j)$ enumerations. Note, the $CPC_i(j)$s generated for a $CP_i(k)$ involves all cuts of the network and, hence, are expensive, especially for networks with many cuts. The number of generated $CPC_i(j)$s for the CPE method is at most the same as for Alg2. Refer to the difference between the two total numbers as the 'saving' of CPE method over Alg2. By avoiding generation of some $CPC_i(j)$s, CPE algorithm speeds up the capacity evaluation process of failure $CPC_i(k)$s. Thus, the improvement of CPE over Alg2 is obvious.

The reliability values (CRRs) are obtained using CAREL [4]. From these reliability values we observe that there is tradeoff between allowing larger flow in the network (make W_{min} larger) and the reliability of the network to carry out that flow. And for certain values of W_{min} we have a good bargain. For example, in table 4a, increasing the W_{min} from 7 to 12 units (71.4% increase) reduces the reliability of the network by only 0.038082 (3.9%). If we increase the W_{min} from 8 to 10 units, the reliability of the network is unchanged, which is favorable for the user of the network. On the other hand, for

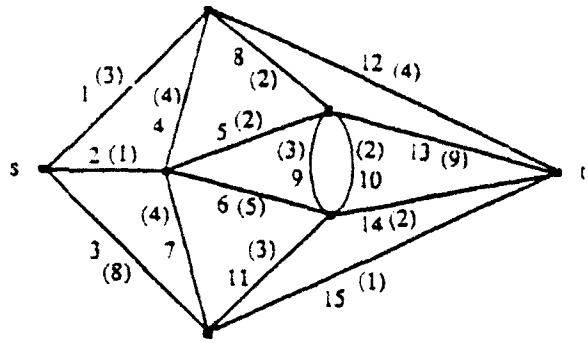


Figure 5. 7-Node, 15-Link Network
[The link capacity is shown within ( ).]

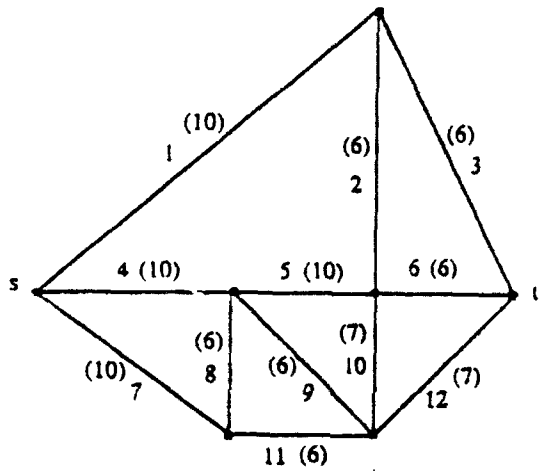


Figure 6. 7-Node, 12-Link Network
[The link capacity is shown within ( ).]

some cases, increasing the W_{max} by 1 unit may reduce the reliability of the network appreciably, which is of course unfavorable. For example, in table 4a, increasing W_{max} from 13 to 14 units (7.7%) reduces the reliability of the network by 0.153481 (18.2%). By utilizing the method presented in this paper, we can get the best W_{max} for certain network configurations. Thus, the technique provides a novel approach for comparing the reliability of various candidate topologies having heterogeneous link-capacities.

APPENDIX A

CPE Implementation

Data representation markedly affects the efficiency of algorithms. Since the CPE algorithm uses a lot of set theoretic operations (such as union, intersection, XOR), the bit vector data representation [21] is the best for its implementation. The bit vector operation minimizes the memory used and helps the program run efficiently. A path (cut) state in a network with l links requires only $\text{liub}(l/w) \times w$ bits of memory, for a word size w. Considering $w = 16$, a path $P_i = (2,4,6,12,17)$ is represented as:

word 1 *word* 2

0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
msb *lsb* *msb* *lsb*

For l-bit representation, decoding/encoding of data usually requires l tests of single bits as the worst case. Since in general not all the l bits are set, eventually we might have better performance, especially when most of the set bits are in the same words. The following subsections describe the details of the steps listed in the CPE algorithm (Section 4.1)

A.1 Read Files: Decode the input files into the bit vector representation. We want to associate a path with a certain name $P_i (i = 1,\ldots,m)$, and arrange the whole paths in a path list. The capacity of the links can be kept in an array of integer, *capacity*[L], for $L = 1,2,\ldots,L$

A.2 W_{max} Computation: When $W_{max} < W_{min}$, the network always fails. The W_{max} is defined as $\min\{W(C_i)\}$ using (1). The worst-case time complexity involved is $O(l \times n)$, where n is the number of cuts.

A.3a Success/Failure determination of paths: It is done as follows —

```
for (all paths P_i) {
  if (path_capacity(P_i) ≥ W_min)
    Append P_i to S_set;
  else
    Append P_i to F_set(1);
}
```

The path _capacity ( ) function will return the capacity of a path P_i. In (2), the capacity of a path is given as $\min(capacity[L_i])$, where L_i's are UP links in that path. The time complexity for A_3a is $O(l \times m)$.

A.3b Getting *key_cut*: The *key_cut* concept reduces the number of states enumerated and, thus reduces the running time of the CPE algorithm. Using the set theoretic operations, finding a *key_cut* is straightforward. The algorithm is —

```
if (key_cut(source node cut C_s) or key_cut(terminal node cut C_t))
  let the min (|C_s|,|C_t|) be the key_cut;
if (C_s and C_t are not key_cut) {
  for (all cuts C_i)
    select as a key_cut the min (|key_cut(C_i)|);
}
```

The function key_cut( ) determines a *key_cut* based on its definition in Section 3.3. The time complexity of the key_cut( ) function is $O(l)$ and, hence, the time complexity to find a *key_cut* is $O(l \times n)$.

A.3c Path_group Generation: When a *key_cut* exists, we intend to reduce the number of states generated by creating *path_groups* based on the *key_links*. The time complexity involved is $O(l \times \beta)$ where β is the number of failure simple paths. It is done as follows —

```
for (i = 1 to l) {
  if (i ∈ key_links) {
    for (all CP_j(1) in F_set(1)) {
      if (i ∈ CP_j(1))
        let CP_j(1) be a member of group G(i);
    }
  }
}
```

A.3d 2-Composite Path Generation: The generation of 2-composite paths directly from $F\_set(1)$ is straightforward. We exhaustively create the combinations of two failure states. This method is easy, but generates many redundant states. Use of this method is suggested when the network does not have a *key_cut*. Nonetheless, a conservative technique to generate the 2-composite paths from the *path_groups* is employed. For a source or terminal node cut working as a *key_cut*, each member of a certain *path_group* is a unique path. And creating 2-composite path is also unique. Hence, we do not have to check for redundant states. On the other hand the members of the *path_groups* generated from the *key_link* of other than the above *key_cuts* is not unique. In such cases, it happens that generated 2-composite paths comprised from the paths which are the member of a certain group, and are always a failure state. To help reduce the time computation, we need to check such cases. The time complexity of this step is $O(\beta^2)$ if *key_cut* does not exist.

A.4 k-composite paths generation ($k>2$): Here, we generate a k-clique from 2 $(k-1)$-cliques of $F\_set(k-1)$ using set theoretic XOR operation. If the result is a link in PG(V,E), they have formed a k-clique. Otherwise the 2 $(k-1)$-cliques fail to produce a k-clique. If the 2 $(k-1)$-cliques create a k-clique, get the k-clique by taking the union of those 2 $(k-1)$-cliques. Note, we may keep the $F\_set(2)$ in lexicographic order to speed up the searching process.

- Success/Failure Determination (for $CP_l(k)$): Lemma 2 is incorporated to determine the success/failure of a $CP_l(k)$. The algorithm is —

```
for (all cuts C_{1,l}) {
  if (W(CPC_l(j)) < W_min) {     /* (1) */
    append it to F_set(k);       /* check for redundancy first */
    break;
  }
  if (CP_l(k) is not a failure)
    append it to S_set;          /* check for redundancy first */
}
```

For the worst case, the method uses all cuts to help determine a success $CP_l(k)$. Thus, the time complexity for each state $CP_l(k)$ is $O(n)$.

- *Cross_link*$_l(k)$ Generation: For each $CP_l(k)$, *cross_link*$_l(k)$ is generated by taking the intersection of *cross_link*$_l(k-1)$ and *cross_link*$_R(k-1)$ of $CP_J(k-1)$ and $CP_R(k-1)$, respectively. Thus, the time complexity for each *cross_link*$_l(k)$ is constant.

APPENDIX B

B.1 A counter example for method [10]

Consider figure 2 where the capacity of each link is the number shown in parentheses. Nine simple paths for the network are: $P_1=(2,5,6)$, $P_2=(3,7,8)$, $P_3=(1,4,6,8)$, $P_4=(5,7)$, $P_5=(1,2)$, $P_6=(4,5,8)$, $P_7=(3,4)$, $P_8=(1,6,7)$, and $P_9=(2,3,6,8)$. All the paths represent failure states for $W_{min}=5$. Now, generate $\binom{9}{2}=36$ 2-composite paths from the failure simple paths. While checking for the successes, one obtains $CP_{2,5}(2)$ as the only success 2-composite paths. Since P_2 and P_5 have yielded success composite paths, [10] eliminates them from the simple path list. Seven remaining simple paths, P_1, P_3, P_4, P_6, P_7, P_8, P_9, are then utilized to generate $\binom{7}{3}=35$ 3-composite paths. Since P_5 and P_2 are no longer in the simple path list, method [10] fails to generate a non-redundant 3-composite path $CP_{1,5,7}(3)$. Note, $CP_{1,5,7}(3)$ is a success 3-composite path (refer to figure 2). Obviously, the method of [10] does not work in general.

B.2 A counter example for method [13]

For the following counter example, we use the notation in [13]. Consider figure 2 and its two typical paths: $P_1=(2,5,6)$, and $P_2=(3,7,8)$ with path capacities $C_1=1$ and $C_2=2$, respectively. Since there is no common link in the two paths, we use the modified step 2 of [13] to evaluate the capacity of the combination of paths P_1 and P_2 (C_{12}). Following [13], we define the vector $V=[0\ 4\ 3\ 0\ 1\ 2\ 2\ 3]$, and initialize $j=1$, and $C_{12}=0$.

for $j=1$, $x_1=1$; and $V=[0\ 3\ 3\ 0\ 0\ 1\ 2\ 3]$.

for $j=2$, $x_2=2$; and $V=[0\ 3\ 1\ 0\ 0\ 1\ 0\ 1]$.

Therefore $C_{12} = 1 + 2 = 3$.

However, the definition of flow in [18] confirms that when all links in paths P_1 and P_2 are good, the links are capable of carrying 4 units of flow. Hence, the modified method in [13] is imperfect too. Note, the proposed method presented in subsection 3.1 gives the correct capacity of 4 units for C_{12}. It is observed that for two or more disjoint paths having at least one node (other than the source or terminal nodes) common amongst themselves, [13] is likely to lead to an incorrect conclusion.

REFERENCES

[1] S. Rai, S. Soh, "Survivability analysis of complex computer-network with heterogeneous link-capacities", *Proc. Ann. Reliability and Maintainability Symp.*, 1991, pp 374–379.

[2] S. Harari, C. S. Raghavendra, "SYREL: A symbolic reliability algorithm based on path and cutset methods", *IEEE Trans. Computers*, vol C-36, 1987 Oct, pp 1224–1232.

[3] C. J. Colbourn, *The Combinatorics of Network Reliability*, 1987; Oxford University Press.
[4] S. Soh, S. Rai, "CAREL: Computer aided reliability evaluator for distributed computer networks", *IEEE Trans. Parallel and Distributed Systems*, vol 2, 1991 Apr, pp 199–213.
[5] S. Rai, D. P. Agrawal, *Distributed Computing Network Reliability*, 1990; IEEE Computer Society Press. (Tutorial Text)
[6] J. M. Wilson, "An improved minimizing algorithm for sum of disjoint product", *IEEE Trans. Reliability*, vol 39, 1990 Apr, pp 42–45.
[7] L. B. Page, J. E. Perry, "Reliability of directed networks using the factoring theorem", *IEEE Trans. Reliability*, vol 38, 1989 Dec, pp 556–562.
[8] P. Doulliez, E. Jamouille, "Transportation networks with random arc capacities", *Revue Francaise d' Automatique, Informatique Recherche Operationnelle*, vol 2, 1972 Nov, pp 45–59.
[9] S. H. Lee, "Reliability evaluation of a flow network", *IEEE Trans. Reliability*, vol R-29, 1980 Apr, pp 24–26.
[10] K. B. Misra, P. Prasad, "Comments on: Reliability evaluation of flow networks", *IEEE Trans. Reliability*, vol R-31, 1982 Jun, pp 174–176.
[11] S. Rai, A. Kumar, E. V. Prasad, "Computing the performance index of a computer network", *Reliability Engineering*, vol 16, 1986, pp 153–161.
[12] K. K. Aggarwal, et al, "Capacity consideration in reliability analysis of communication systems", *IEEE Trans. Reliability*, vol R-31, 1982 Jun, pp 177–181.
[13] K. K. Aggarwal, "A fast algorithm for the performance index of a telecommunication network", *IEEE Trans. Reliability*, vol 37, 1988 Apr, pp 65–69.
[14] W. J. Rueger, "Reliability analysis of networks with capacity constraints and failures at branches and nodes", *IEEE Trans. Reliability*, vol R-35, 1986 Dec, pp 523–528.
[15] R. Sahner, K. Trivedi, "Performance and reliability analysis using directed acyclic graphs", *IEEE Trans. Software Engineering*, vol SE-13, 1987 Oct, pp 1105–1114.
[16] M. O. Locks, "A minimizing algorithm for the sum of disjoint products", *IEEE Trans. Reliability*, vol R-36, 1987 Oct, pp 445–453.
[17] M. O. Locks, "Inverting and minimizing pathsets and cutsets", *IEEE Trans. Reliability*, vol R-27, 1978 Jun, pp 107–109.
[18] L. R. Ford, D. R. Fulkerson, *Flows in Networks*, 1962; Princeton University Press.
[19] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, 1980; Academic Press.
[20] K. V. Le, V. O. K. Li, "A path-based approach for analyzing reliability of systems with dependent failures and multimode components", *INFOCOM 90*, pp 495–503.
[21] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, 1974; Addison Wesley

AUTHORS

Dr. Suresh Rai; Department of Electrical and Computer Engineering; Louisiana State University; Baton Rouge, Louisiana 70803-5901 USA.

Suresh Rai (SM'86) is an Associate Professor with the Department of Electrical and Computer Engineering at Louisiana State University, Baton Rouge. Dr. Rai has taught and researched in the area of reliability engineering, fault diagnosis, and parallel and distributed processing. He is a co-author of the book *Waveshaping and Digital Circuits*, and tutorial texts *Distributed Computing Network Reliability and Advances in Distributed System Reliability*. He has guest edited a special issue of *IEEE Transactions on Reliability* on the topic *Reliability of Parallel and Distributed Computing Networks*. Dr. Rai has worked as program committee member for IPCCC '91 (Phoenix) conference. Currently, he is an Associate Editor for this *Transactions*. Dr. Rai received his BE from Benaras Hindu University in 1972, his ME from University of Roorkee in 1974, and his PhD from Kurukshetra University in 1980. He joined the Regional Engineering College at Kurukshetra in 1974 and worked there for six years. After a brief stay at MMM Engineering College, Gorakhpur, he transferred to the University of Roorkee in 1981 as an Associate Professor. He also worked for two years at the School of Engineering at North Carolina State University, Raleigh. Dr. Rai is a senior member of the IEEE, and member of the ACM.

Sieteng Soh; Department of Electrical and Computer Engineering; Louisiana State University; Baton Rouge, Louisiana 70803-5901 USA.

Sieteng Soh (SM'91) received the BS in Electrical Engineering (Computer option) in 1986 from the University of Wisconsin, Madison, and the MS in Electrical Engineering in 1988 from Louisiana State University, Baton Rouge. He is a PhD student in the Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge. His research interests include computer architecture, fault tolerant computing, parallel processing and reliability of digital systems. Mr. Soh is a student member of the IEEE, Eta Kappa Nu, and Tau Beta Pi.

Manuscript TR91-309 received 1990 November 15; revised 1991 March 8.

IEEE Log Number 02221 ◄TR►

MANUSCRIPTS RECEIVED *MANUSCRIPTS RECEIVED* *MANUSCRIPT RECEIVED* *MANUSCRIPTS RECEIVED*

"Some paradoxes of non-homogeneous Poisson process models in software reliability" (Xie, Zhao), Dr. Min Xie ⊏ Dept. of Quality Technology ⊐ Linkoping Institute of Technology ⊏ S-581 83 Linkoping ⊐ SWEDEN. (TR91-123)

"The general linear regression analysis applied to a new 3-parameter Weibull distribution" (Li), Li, Yong-Ming ⊏ POBox 1501-774 ⊏ Guangzhou ⊏ Peop. Rep. CHINA. (TR91-124)

"Reliability of a standby system with beta-distributed component lives" (Pham, Turkkan), Dr. T. G. Pham ⊏ Faculty of Science & Engineering ⊏ Universite de Moncton ⊏ Moncton, New Brunswick E1A 3E9 ⊏ CANADA. (TR91-125)

"A connectionist model for the realization of Markov transition diagrams" (Suliman), Mamoun M. A. Suliman ⊏ POBox 2145 ⊏ Carbondale, Illinois 62902-2145 ⊏ USA. (TR91-126)

"Prediction for life data from exponential distribution: An averaging approach" (Wong), Augustine C. M. Wong ⊏ Dept. of Statistics & Actuarial Science ⊏ University of Waterloo ⊏ Waterloo, Ontario N2L 3G1 ⊏ CANADA. (TR91-127)

"Reliability aspects of memory-management policies" (Bowen, Pradhan), Nicholas S. Bowen ⊐ IBM T.J. Watson Research Center ⊏ POBox 704, H4-C12 ⊏ Yorktown Heights, New York 10598 ⊐ USA. (TR91-128)

"Some characteristic properties of the generalized hyperexponential distribution" (Nassar), Manal Mohamed Nassar, Professor ⊏ Dept. of Mathematics ⊐ Ain Shams University ⊏ Abbassia, Cairo ⊏ EGYPT. (TR91-129)

"Failure mechanisms of materials" (Dasgupta, Pecht), Dr. Michael G. Pecht, PE ⊏ CALCE Center for Electronic Packaging ⊐ University of Maryland ⊐ College Park, Maryland 20742 ⊏ USA. (TR91-130)

"A fuzzy-set approach to software-reliability modeling" (Mihalache), Adrian N. Mihalache ⊐ bd M. Kogalniceanu 61 ⊐ Bucharest ⊐ ROMANIA. (TR91-131)

Experimental Results on Preprocessing of Path/Cut Terms in Sum of Disjoint Products Technique

Sieteng Soh and Suresh Rai

Department of Electrical & Computer Engineering
Louisiana State University, Baton Rouge, LA 70803

Abstract - In the sum of disjoint products (SDP) methods of network reliability analysis, the *preprocessing* of minimal paths/cuts is necessary to help reduce the number of SDP terms and, hence, the overall computation time. Several researchers have proposed 1) cardinality-, 2) lexicographic-, and 3) Hamming distance [18]- ordering methods to preprocess the path terms in the SDP techniques. For cutsets, an ordering based on the node partition associated with each cut is suggested [3]. Our paper presents experimental results showing the number of disjoint products and computer time involved in generating SDP terms. To help obtain the results, we have considered 19 benchmark networks containing paths (cuts) varying from 4 (4) to 780 (7376). Several SDP techniques are reviewed and are generalized into three propositions to find their inherent merits and demerits. An efficient SDP technique is, then, utilized to run input files of paths/cuts preprocessed using 1) through 3) and their combinations. The experimental evaluation has been performed on an FPS 500 system. Finally, results are analyzed, and it is shown that the preprocessing based on cardinality or its combinations with 2) and/or 3) performs better.

I. Introduction

Several algorithms dealing with the terminal reliability evaluation are proposed in the literature. These methods can be classified as: state enumeration, decomposition technique, inclusion - exclusion, factoring, and sum of disjoint products. A summary of these techniques, including their relative merits and demerits, can be found in [1]. Note, all methods of terminal reliability computation are known to be NP-hard [2].

The sum of products technique (SPT) utilizes Boolean concepts to change a path or cut polynomial into an equivalent sum of disjoint products (SDP) expression and, thus, generates the reliability parameter of a given network. Algorithms based on this principle start with a Boolean polynomial formed by either the success terms (minimal paths) or the failure terms (minimal cuts) for a given network. To help reduce the generated SDP terms, the paths or cuts are sequenced according to the cardinality [5,15], and/or lexicographic ordering following the symbols of their alphabets [6]. Recently, Wilson [4] has suggested to choose the first term in a group lexicographically and, then, sort the second and subsequent terms so that the number of variables a term has in common with the preceeding term is maximized. Buzacott [3] proposes an alternative approach to order the minimal cuts based on the node partition associated with each cut. An SPT, then, converts this preprocessed Boolean polynomial of paths/cuts into an equivalent SDP form that represents the disjoint system logic [1-2,5,7-11,13,15-17]. The disjoint products for any (m-1) size path P_i, where m denotes the number of nodes in the graph model G(V,E), is obtained directly by intersecting the complements of the remaining {l-(m-1)} links of G(V,E) with P_i. This observation (first made in [5] and, then, proved in [10]) further reduces the computation time for algorithms based on the Boolean concepts. Note, an SDP expression has one to one correspondence with the system probability formula. A drawback of the algorithms based on the manipulation of Boolean sum of products or implicants is in the iterative application of certain operations and the fact that the Boolean function changes at every step and may be clumsy. Moreover, the Boolean function is simplified using absorption rules [18] and, thus, requires a considerable computational effort [8]. Therefore, most SPTs are applicable only to small to moderate sized networks. Recently, Soh and Rai [7] have proposed CAREL (Computer Aided RELiability evaluator), a new algorithm based on SPT concept, which can evaluate a large network (with 780 paths and 7376 cuts) in less than a minute CPU time (on Encore MULTIMAX system) with modest memory requirement.

All preprocessing techniques reduce the number of SDP terms, hence, the computation time of reliability analysis; however, there is no unified study or experimental work on their comparative performances. This paper provides experimental results to help compare their performances. For this, we have used 19 small to large benchmark networks with paths (cuts) from 4 (4) to 780 (7376). We consider 1) cardinality- , 2) lexicographic- , 3) Hamming distance- ordering methods and their combinations to preprocess paths/cuts for the benchmark networks. The CAREL [7] is, then, utilized to obtain the SDP terms and

This work was supported in part by the Air Force Office of Scientific Research under grant AFOSR-90-0324.

CH2979-3/91/0000-0533 $1.00 © 1991 IEEE

computation time for generating these terms. [In fact, any existing Boolean technique may be used to perform the experiment.] The experimental evaluation is performed on an FPS 500 system.

The layout of the paper is as follows : Section II presents preliminaries. It also discusses bit vector data representation technique which has been used to program CAREL [7] and sorting methods utilized in preprocessing. Section III considers a generalized view of SPTs and compares their basic philosophies. An example is solved to illustrate their concepts. Section IV discusses preprocessing methods and provides the experimental results using 19 benchmark networks. Note, the benchmark networks are a good representative of small to large graphs because they contain paths (cuts) varying from 4 (4) to 780 (7376). Finally, Section V outlines a discussion on the experimental results.

II. Preliminaries

2.1 *Background*

Consider a linear graph G(V,E) model for a network where V denotes the set of nodes and E represents the set of edges of the network. Note, $|V| = n$, and $|E| = l$. Assume G(V,E) is free from self loops and directed cycles. Each edge has two states : good (UP) or bad (DOWN). Nodes are assumed to be perfect (imperfect nodes can be considered following a method given in [5]). Let the link failures be statistically independent. This assumption is useful to make the reliability problem mathematically tractable [1].

A minpath P_i is a path from a source node s to a terminal node t in G(V,E). It is formed by the set of UP edges such that no nodes are traversed more than once. Pathset is defined as the set of minpaths. A cut is a disconnecting set. All communication between a prescribed (s,t) node pair is disrupted once the edges in (s,t) cut fail. We define a mincut as a cut which has no proper subset that is also a cut, and cutset as the set of mincuts. Assume that either pathset or cutset between a source s and terminal t in G(V,E) is known.

2.2 *Data Representation*

Data structure is an important aspect of designing efficient algorithms [12]. Rosenthal [14] has discussed the advantage and disadvantage of three different kinds of data representation. This section describes one of the representations, namely bit vector representation, which can be used efficiently on some of the Boolean SDP techniques [7,10] and also for lexicographic or Hamming distance [18] ordering discussed later in the text.

A minpath/mincut in a network with l links is represented by an *identifier* having l bits. An UP link of the network is denoted by a binary 1. A binary 0 stands for a *don't care* state (not a DOWN state). Consider the minpaths *a b, c d, a d e and b c e* between the (s,t) node pair in Figure 1. These minpaths are stored in the memory as following identifiers : (Leftmost bit is the most significant bit.)

a b : 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
c d : 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
a d e : 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
b c e : 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0

In this example, we have utilized the word size w as 16 bits. Note, a minpath/ mincut requires $\lceil l/w \rceil$ words of memory. With bit vector representation, the storage requirement for a minpath (mincut) identifier depends on the total number of links in the network and not on the size of the pathset (cutset). Coding and decoding of path information into bit representation or identifier and vice versa may add extra cost as it involves l bits testing. However, this operation (coding and decoding) of minpaths/ mincuts are one time operation. They are usually worth the extra computation as the generation of disjoint events (and, hence, the reliability) requires considerable manipulations. Moreover, the ability of bit representation in detecting and eliminating redundant terms using set theoretic operations like union, intersection, subset etc., is an important advantage. To illustrate the concept for redundancy checking, assume the reference term as X and a test term XY (which is a redundant subset of X). Then do the following:

```
reference (X)      1 1 0 0 1
test      (XY)     1 1 1 0 1
                   ---------
                   1 1 1 0 1      ;OR operation
test      (XY)     1 1 1 0 1
                   ---------
                   0 0 0 0 0      ;EOR operation
```

A result '0 0 0 0 0' shows XY as redundant. Similarly, a duplicate term is detected and deleted. Thus, the set theoretic operations are easy to implement. Note, the computation time is independent of the size of the network. The number of links l, which affects the speed of the set operations, increases the computation time by one unit for every w additional links.

The bit vector representation can be used efficiently for lexicographic sorting. Since the alphabets of our symbols are mapped into bit representation from most to least significant bit, the lexicographic sorting can be achieved simply by sorting the integer representation of the paths/cuts in decreasing order. Note, the Hamming distance [18] between two paths (cuts) is defined as the number of bits in which their identifiers differ. We have also used bit vector representation to implement Hamming distance ordering.

III. Sum of Disjoint Products Technique

3.1 Boolean Techniques Concept [7]

Boolean techniques of reliability evaluation start with a sum of products expression for pathsets or cutsets and convert it into an equivalent sum of disjoint products (SDP) expression. In the SDP form, an UP or logical success (DOWN or failure) state of a link x is replaced by link reliability p (unreliability q), and the Boolean sum (product) by the arithmetic sum (product). In other words, the SDP expression is interpreted directly as an equivalent probability expression of terminal reliability. If F_i represents a path identifier (an UP state of a link in a path P_i has 1 in F_i, while a *don't care* is represented by 0), the sum of products expression F is given by :

$$F = \bigcup_{i=1}^{n} F_i \qquad (1)$$

where n denotes the number of minpaths (mincuts) between (s,t) node pair in G(V,E). Note, F_i's in (1) are preprocessed (ordered) following any methods in [3-6]. Equation (1) is modified either canonically or conservatively to generate the equivalent SDP expression, F(disjoint). The conservative modification is usually preferred, since it is more efficient compared with canonical modification, where 2^l events are required to determine F(disjoint). [l is the number of links in the network.] A simple way to generate the mutually disjoint events in (1) is as follows:

$$F_1 + F_2\overline{F_1} + F_3\overline{F_1}\,\overline{F_2} + \ldots + F_n\overline{F_1}\,\overline{F_2} \ldots \overline{F_{n-1}}$$

where $\overline{F_i}$ denotes DOWN events of path P_i. The probability of UP (operational) for an *ith* term $F_i\,\overline{F_1}\,\overline{F_2} \ldots \overline{F_{i-1}}$ can be evaluated using conditional probability and standard Boolean operations as :

$$\Pr(F_i) \cdot \Pr(\overline{F_1} \cdot \overline{F_2} \ldots \overline{F_{i-1}} \mid F_i) = \Pr(F_i) \prod_{j=1}^{i-1} \Pr(E_j)$$

Here, an E_j represents a conditional cube [10] and defines conditions for a path identifier F_j DOWN given F_i UP (operational). The probability of the first event $\Pr(F_i)$ can be determined in a straightforward manner since the failures are assumed to be statistically independent. However, the coefficient $\Pr(E_j)$ requires further consideration since various terms within E_j's will, in general, be *not* disjoint [1]. This necessitates E_j's to be made mutually disjoint before we generate the equivalent probability expression.

Various researchers [1] have worked on this philosophy and have proposed methods to generate a disjoint expression for (F_i, F_k) pairs in (1), and also the (E_i, E_j) terms within an F_i. Following three propositions P I through P III that convert F into F(disjoint) represent basic principles behind most Boolean SDP methods in the literature :

Table 1.
OP_1 through OP_3 used with SDP techniques

Operator	Function	Ref.
OP_2	Cutset Disjoint Procedure	[10]
OP_3	$ operator	[11]
OP_2	E-operator	[5]
OP_3	COMPARE( ) function	[15]
OP_1	CMB (*) operator	[9]
OP_3	Boolean negation	[17]
OP_2	Relative complement, Procedure 1	[13]
$OP_{1.}$	method 1, CMB (*) operator	[7]
OP_2	method 2, CMB (*) operator	[7]

Proposition P I. The proposition P I defines intermediate term(s) T_i's as :

$$T_i = \bigcup_{j=1}^{i-1} F^j \mid_{\text{each literal of } F_i \to 1} \qquad (2)$$

where $F^1 = F_1$ and $F^j = F_j\ OP_1\ T_j$. Here, F^j refers to the equivalent disjoint product term(s) for F_j. The operation 'OP_1' is a necessary disjointing operator. (Table 1 lists various operators). The F(disjoint) expression is, then, given by :

$$F(disjoint) = \bigcup_{j} F^j \qquad (3)$$

Algorithms [9] and method 1 in [7] make use of proposition P I.

Proposition P II. For each term F_i, $1 < i \le n$, T_i is defined to be the union of all predecessor terms $F_1, F_2, \ldots, F_{i-1}$, in which any literal that is present in both F_i and any of the predecessor terms is deleted from those predecessor terms, i.e. ,

$$T_i = \bigcup_{j=1}^{i-1} F_j \mid_{\text{each literal of } F_i \to 1} \qquad (4)$$

Consider $F^1 = F_1$, and define $F^j = F_j\ OP_2\ T_j$. Equation (3), then, obtains the equivalent F(disjoint) expression. Hariri and Raghavendra [10], Rai and Aggarwal [5], Bennetts [13], and method 2 in Soh and Rai [7] have based their techniques on proposition P II.

Proposition P III. For $1 < j \le n$, use operation 'OP_3' to perform :

$$F^j = (\ldots((F_j\ OP_3\ F_1)\ OP_3\ F_2)\ OP_3 \ldots)\ OP_3\ F_{j-1} \qquad (5)$$

Equation (5) obtains a set of disjoint cubes [11] corresponding to F_j. Note, $F^1 = F_1$, and OP_3 represents an appropriate disjointing operator. The F(disjoint)

expression is, then, given by equation (3). Abraham [15], Grnarov et.al. [11], and Tiwari and Verma [17] have proposed their methods using P III concept.

Example To illustrate propositions P I through P III, consider a network shown in Figure 4. For the (s,t) node pair, there exists 13 minpaths : *adh, beh, bfi, aceh, acfi, adgi, bcdh, begi, bfgh, acegi, acfgh, adefi,* and *bcdgi*. In what follows, we explain steps to generate the exclusive and mutually disjoint event(s) or cube(s) for F_4 (= *aceh*). This is demonstrated using typical methods for propositions P I through P III. For uniformity, we keep the notation given in [5], in which an $\bar{x}$ denotes the down state of the link x.

P I : Considering CAREL method 1 [7], $F^1 = adh$, $F^2 = beh\ \bar{a}\bar{d}$, and $F^3 = bfi\ (\bar{h} + h\ \bar{e}\ \bar{a}\bar{d})$. Use equation (2) to define T_4 as $(d + b)$ for F_4. Replacing OP_1 with CMB operator [7], we get CMB(T_4) as $\bar{d}\ \bar{b}$. F^4 is then obtained as : $F^4 = F_4$ CMB (T_4). For $j = 5, \ldots, 13$, the terms F^j's, are generated similarly. An expression for F(disjoint) is :

$$\begin{aligned}\text{F(disjoint)} = {} & adh + beh(\bar{a}\bar{d}) + bfi(\bar{h} + h\bar{e}\ \bar{a}\bar{d}) \\ & + aceh(\bar{d}\ \bar{b}) + acfi(\bar{b}\ \bar{h} + \bar{b}h\bar{d}\ \bar{e}) \\ & + adgi(\bar{h}\ \bar{f} + \bar{h}f\bar{b}\ \bar{e}) + bcdh(\bar{a}\ \bar{e}\ \bar{f}i) \\ & + begi(\bar{h}\ \bar{f}\ \bar{a}\bar{d}) + bfgh(\bar{e}\ \bar{i}\ \bar{d} + \bar{e}\ \bar{i}\ d\ \bar{a}\ \bar{c}) \\ & + acegi(\bar{h}\ \bar{f}\ \bar{d}\ \bar{b}) + acfgh(\bar{b}\ \bar{d}\ \bar{e}\ \bar{i}) \\ & + adefi(\bar{b}\ \bar{c}\ \bar{g}\ \bar{h}) + bcdgi(\bar{a}\ \bar{e}\ \bar{f}\ \bar{h}).\end{aligned}$$

P II : We have used E-operator [5] to explain the operation OP_2 and, hence, the concept behind proposition P II. The terms $F^1 = adh$, $F^2 = beh$ $(\bar{a} + a\bar{d})$, and $F^3 = bfi$ $(\bar{h} + h\ \bar{a}\ \bar{e} + ha\ \bar{d}\ \bar{e})$ are computed using [5]. To generate F^4, an intermediate term T_4 is obtained as :

$T_4 = adh + beh + bfi\ |_{\text{exclusive-sharp}} = d + b + bfi$, where the last term is redundant and can be deleted. Substituting OP_2 with E-operator, E (T_4) is $\bar{d}\ \bar{b}$. Hence, $F^4 = F_4$ E (T_4). Similarly, we obtain F^j's for $j = 5, \ldots, 13$. Equation (3) is :

$$\begin{aligned}\text{F(disjoint)} = {} & adh + beh(\bar{a}+a\bar{d}) + bfi(\bar{h}+h\ \bar{a}\ \bar{e}+ha\ \bar{d}\ \bar{e}) \\ & + aceh(\bar{b}\ \bar{d}) + acfi(\bar{b}\ \bar{h} + \bar{b}\ h\ \bar{d}\ \bar{e}) \\ & + adgi(\bar{h}\ \bar{f}+\bar{h}\ f\ \bar{b}\ \bar{e}) + bcdh(\bar{a}\ \bar{e}\ \bar{f}+\bar{a}\ \bar{e}fi) \\ & + begi(\bar{h}\ \bar{f}\ \bar{a}+\bar{h}\ \bar{f}a\bar{d}) + bfgh(\bar{e}\ \bar{i}\ \bar{d}+\bar{e}\ \bar{i}d\bar{a}\ \bar{c}) \\ & + acegi(\bar{h}\ \bar{f}\ \bar{d}\ \bar{b}) + acfgh(\bar{b}\ \bar{d}\ \bar{e}\ \bar{i}) \\ & + adefi(\bar{b}\ \bar{c}\ \bar{g}\ \bar{h}) + bcdgi(\bar{a}\ \bar{e}\ \bar{f}\ \bar{h}).\end{aligned}$$

P III : Use reference [15] to obtain the terms $F^1 = adh$, $F^2 = beh$ $(\bar{a} + a\ \bar{d})$ and $F^3 = bfi$ $(\bar{a}\ \bar{e} + \bar{a}\ e\ \bar{h} + a\ \bar{d}\ \bar{e} + a\ \bar{d}\ e\ \bar{h} + ad\ \bar{h})$. Here, the COMPARE function [15] substitutes OP_3. Then,

$F^4 = ((F_4\ OP_3\ F_1)\ OP_3\ F_2)\ OP_3\ F_3)$.

The inner term $F_4\ OP_3\ F_1$ gives *aceh* $\bar{d}$, which with F_2 generates *aceh* $\bar{d}\ \bar{b}$. Finally, the OP_3 for the outer term gives the result as *aceh* $\bar{d}\ \bar{b}$. Similarly, compute F^j for F_j ($j = 5, \ldots, 13$). Equation (3), then, gives :

$$\begin{aligned}\text{F(disjoint)} = {} & adh + beh(\bar{a}+a\ \bar{d}) + bfi(\bar{a}\ \bar{e}+\bar{a}e\bar{h}+a\bar{d}\ \bar{e} \\ & +a\ \bar{d}\ e\ \bar{h}+ad\ \bar{h}) + aceh(\bar{d}\ \bar{b}) + acfi(\bar{d}\ \bar{b}\ \bar{e} \\ & +\bar{d}\ \bar{b}\ e\ \bar{h}+d\bar{h}\ \bar{b}) + adgi(\bar{h}\ \bar{b}\ \bar{c}+\bar{h}\ \bar{b}\ c\ \bar{f}+\bar{h}b\bar{f}) \\ & + bcdh(\bar{e}\ \bar{a}\ \bar{f}+\bar{e}\ \bar{a}fi) + begi(\bar{a}\ \bar{h}\ \bar{f}+a\ \bar{d}\ \bar{h}\ \bar{f}) \\ & + bfgh(\bar{a}\ \bar{e}\ \bar{i}\ \bar{c}+\bar{a}\ \bar{e}\ \bar{i}c\bar{d}+a\bar{d}\ \bar{e}\ \bar{i}) \\ & + acegi(\bar{d}\ \bar{b}\ \bar{h}\ \bar{f}) + acfgh(\bar{d}\ \bar{b}\ \bar{e}\ \bar{i}) \\ & + adefi(\bar{h}\ \bar{b}\ \bar{c}\ \bar{g}) + bcdgi(\bar{a}\ \bar{e}\ \bar{f}\ \bar{h}).\end{aligned}$$

Note, F(disjoint) expression obtained from different propositions when expanded out should be identical. For the network, the terminal reliability is 0.977184 when each link is assumed to have a reliability of 0.9.

3.2 Existing SDP Techniques - A Comparison [7]

Propositions P I through P III maintain the minpaths or mincuts list in memory (equation (1)) . Consider 1 for UP link and 0 for *don't care*, and utilize bit representation technique [discussed in Section 2.2]. The memory requirement is, then, $\lceil l/w \rceil$ words per path (cut), where l is the number of links in the network G(V,E). Proposition P I makes F_i disjoint with respect to $\bigcup_{j=1}^{i-1} F^j$, while propositions P II and P III utilize $\bigcup_{j=1}^{i-1} F_j$. Should we have similar operations to implement equations (2) and (4), the proposition P I will require more operations than that needed for P II. Generally, an F_i generates more than one SDP terms F^j's. Hence, the number of terms involved in $\bigcup_j F^j$ is larger than that in $\bigcup_j F_j$. For example, Table 4 (N_{780}^{2776}) shows results for i = 780. The number of terms in equation (2) is more than 50,000 ; on the other hand equation (4) needs exactly (i-1) i.e., 779 terms. Note, in proposition P I, the generated SDP terms have to be kept in the memory to implement equation (2), which is not the case for P II or P III. This makes proposition P I sequential. Moreover, P I demands a huge memory space to evaluate a large network. On the other hand P II or P III has implicit parallelism, making it easier for the programmers to implement them on parallel systems. Overall, the proposition P II or P III provides advantages in comparison with P I.

An analysis of performance comparison between a typical example of proposition P II and P III is discussed in [10]. SYREL [10], an implementation technique for E-operator [5], is shown to have better performance in comparison with $-operator [11]. It means proposition P II outperforms P III. Moreover, proposition P II offers a faster implementation approach than that in P I or P III. The bit vector implementation of $\bigcup_j F_j$ makes the realization of equation (4) w (word size) times faster than generating equation (2) based on proposition P I. In what follows, we use CAREL [7] to perform our experiment.

Table 2.
Preprocessing results for paths in Figure 4

Random	H	L	C	C+H	C+L
adh	*adh*	*acegi*	*bfi*	*adh*	*adh*
adgi	*bcdh*	*aceh*	*beh*	*beh*	*beh*
adefi	*acfgh*	*acfgh*	*adh*	*bfi*	*bfi*
aceh	*aceh*	*acfi*	*bfgh*	*adgi*	*aceh*
acegi	*adefi*	*adefi*	*begi*	*acfi*	*acfi*
acfgh	*adgi*	*adgi*	*bcdh*	*begi*	*adgi*
acfi	*bfgh*	*adh*	*acfi*	*aceh*	*bcdh*
bcdh	*beh*	*bcdgi*	*aceh*	*bcdh*	*begi*
bcdgi	*bcdgi*	*bcdh*	*adgi*	*begh*	*bfgh*
beh	*acfi*	*begi*	*bcdgi*	*adefi*	*acegi*
begi	*acegi*	*beh*	*acfgh*	*acegi*	*acegh*
bfgh	*bfi*	*bfgh*	*acegi*	*acfgh*	*adefi*
bfi	*begi*	*bfi*	*adefi*	*bcdgi*	*bcdgi*

C = Cardinality ordering
H = Hamming distance ordering
L = Lexicographic ordering

IV. Preprocessing of Path / Cut terms

Various researchers [1] have pointed out the need for preprocessing (ordering) of paths/cuts to help reduce the number of generated SDP terms, and hence, the computation time. References [5,15] sequence the minimal paths in the order of their increasing cardinality. For each group of terms of the same size, Locks [6], further, suggests to sort the terms lexicographically following the order of alphabets used to represent the paths/cuts. Recently, Wilson [4] has modified the preprocessing [6] by incorporating *distance* concept. The *distance*, here, means the number of variables a term has in common with a reference term which is chosen within a group lexicographically. Logically, we may call the *distance* as Hamming distance [18] if we have binary representation for two terms in question.

4.1 Preprocessing Methods

This section presents the concept of various preprocessing methods which are used to order path/cut terms in an SPT. We, primarily, utilize the following ordering approaches to generate our experimental results :

1) increasing Hamming distance,
2) lexicographic,
3) increasing cardinality [5,15],
4) increasing cardinality and lexicographic [6], and
5) increasing cardinality and Hamming distance.

Note, approach 5) is similar to the one suggested in [4] except for the fact that we pick the reference term in a group randomly. Reference [4], on the other hand, chooses a term based on lexicographic ordering.

Table 2 presents the minimal paths of the network shown in Figure 4, and also the results of preprocessing the paths based on the methods 1) through 5). Initially, paths generated for the network are in random order as shown in column 1 of Table 2. For the Hamming distance ordering, we arbitrarily pick the first term as the reference. Then, we sort the paths based on the increasing Hamming distance from the reference. The terms which have the same Hamming distance can be in any order among themselves. The result for this scenario is shown in column 2 of Table 2.

We use alphabets (1,2, .. , *l*) to denote a path/cut of a network, where *l* denotes the number of links in the network. Label each edge with a distinct alphabet. Then, sort the paths lexicographically [12]. The result of this type of preprocessing for the paths in Figure 4 is presented in column 3 of Table 2. The ordering using cardinality is mentioned in [5,15]. Note, the terms with the same cardinality are ordered in any sequence (refer to column 4 of Table 2).

Column 5 of Table 2 shows the result of preprocessing the paths of Figure 4 using the combination of cardinality- and Hamming distance- orderings. For this approach, terms of the same size are sorted out based on Hamming distance from the first term in the group. Note, the reference (or the first term) in a group is chosen randomly. It differentiates our method from [4], where the first term is chosen by lexicographic ordering. Our ordering method is computationally simpler than that in [4] and it does not hinder SPT concept. [The preprocessing time should be less than 5-10 percent of the overall reliability computation time.]

Finally, we consider to preprocess the paths based on both the cardinality- and the lexicographic- ordering [6]. We, first, order the terms in their increasing cardinality. Then, for the terms within the same cardinality group, use lexicographic sorting. The result of this preprocessing is provided in the last column of Table 2.

4.2 Experimental Results

Once we have preprocessed the paths/cuts of a network using approaches 1) through 5) mentioned in subsection 4.1, we generate the SDP terms (and the terminal reliability for the network) using an SPT CAREL [7]. Table 3 lists the 19 benchmark networks that we have utilized for our experiments. The table also illustrates reliability (unreliability) values assuming that each network has link reliability (unreliability) of 0.9 (0.1). We denote each of the benchmarks with notation N_j^i, where the subscript j (superscript i) represents the total number of minpaths (mincuts) for the network. Note, j (i) varies from 4 (4) to

Table 3.
Benchmark Networks used in experiments

No.	Network*	$R(G)$**	$Q(G)$**	Comments
1	N_4^4	0.978480	0.021520	Fig. 1 Bridge network
2	N_7^9	0.968425	0.031575	Fig. 2 6-node, 8-link network
3	N_9^8	0.997632	0.002368	Fig. 3 5-node, 8-link network
4	N_{13}^9	0.977184	0.022816	Fig. 4 Modified ARPANET
5	N_{15}^{28}	0.964855	0.035145	Fig. 5 ARPANET in 1971
6	N_{14}^{15}	0.996665	0.003336	Fig. 6 7-node, 15-link network
7	N_{18}^{110}	0.994076	0.005924	Fig. 7 11-node, 21-link network
8	N_{18}^{24}	0.969112	0.030888	Fig. 8 9-node, 13-link network
9	N_{24}^{19}	0.975116	0.024884	Fig. 9 Modified ARPANET
10	N_{20}^{20}	0.984068	0.015932	Fig. 10 Fig.9 with different (s,t)
11	N_{23}^{20}	0.997494	0.002506	Fig. 11 7-node, 12-link network
12	N_{29}^{29}	0.996217	0.003783	Fig. 12 8-node, 13-link network
13	N_{36}^{396}	0.997186	0.002814	Fig. 13 16-node, 30-link network
14	N_{44}^{328}	0.904577	0.095423	Fig. 14 ARPANET
15	N_{44}^{25}	0.974145	0.025855	Fig. 15 reduced form of Fig.14
16	N_{64}^{78}	0.997506	0.002494	Fig. 16 10-node, 21-link network
17	N_{281}^{1300}	0.985928	0.014072	Fig. 17 ARPANET
18	N_{281}^{214}	0.987390	0.012610	Fig. 18 reduced form of Fig. 17
19	N_{780}^{7376}	0.997120	0.002880	Fig. 19 20-node, 30-link network

* N_i^j means a network with j paths and i cuts
** Terminal reliability (unreliability) values are for link reliability (unreliability) = 0.9 (0.1)

780 (7376). We generate the reliability values of the network from the pathsets, and the unreliability results from the cutsets. As expected the sum of the reliability and unreliability values for a network is 1.

Table 4 presents the experimental results by running CAREL [7] with random and preprocessed lists of pathsets for the benchmark networks. The table provides the results in terms of total number of disjoint paths and the computer time involved in generating SDP terms. Note, the computer time is in CPU seconds, and the '0' second represents a time less than 0.1 second. These results are generated using an FPS 500 system.

Table 5 is obtained similarly. It shows the experimental evaluations for random and preprocessed lists of cutsets. To help evaluate the efficiency of a preprocessing technique, we have used the number of disjoint cuts and computer time consumed to generate them for each benchmark network.

V. Discussion

Tables 4 and 5 suggest that Hamming distance and lexicographic preprocessings do not reduce the complexity of SPT for network reliability evaluation. Both types of preprocessing may or may not reduce the number of disjoint paths/cuts terms and also the CPU time compared to the ones involved for the randomly ordered path/cut terms. The basic idea in having a Hamming distance- or a lexicographic- ordering is to take advantage of overlapping link variables in the paths. But, for large networks, since there are so many permutations of variables involved in the ordering of the paths/cuts, it is likely that the Hamming distance- and lexicographic- orderings take advantage of the overlap only on the early stages of the computation. For the larger part of the evaluation, we may or may not have a better overlapping variables compared to the random order of paths/cuts. This reasoning explains our results (refer to Tables 4 and 5).

On the other hand, cardinality-based preprocessing (either cardinality ordering only or the combinations with Hamming distance- or lexicographic- ordering) shows significant improvements over the other techniques. For larger networks, it is shown in Tables 4 and 5 that both the number of disjoint paths/ cuts and the CPU time required to compute them are dramatically reduced. The results prove the importance of cardinality-based preprocessing (to help reduce the time complexity of SPT for network reliability problem). Note for path-based SPT, the

cardinality-based preprocessings may also take further advantage of the fact that the disjoint term for a path P_i having $(m-1)$ links is generated simply by intersecting the complements of the remaining $\{l-(m-1)\}$ links of the network with P_i [5,10]. This fact significantly reduces the CPU time of reliability evaluation for large networks. But the method does not apply for non-cardinality based preprocessing techniques. Moreover, this observation is hardly applicable for any preprocessing method used on cut terms.

While comparing the cardinality-based techniques, we notice in Tables 4 and 5 that the number of disjoint paths/cuts and the CPU times required from inputting the preprocessed paths/cuts to CAREL [7] are of the same order. Adding Hamming distance- or lexicographic- orderings may or may not reduce the disjoint paths/cuts generated nor the CPU time consumed to generate them. For large networks, the results can be easily explained from the fact that neither lexicographic- nor Hamming distance-orderings give any benefits over random ordering (as explained earlier). Note in large networks, in general, there are a lot of terms which are of the same cardinality. Thus, ordering the terms in each group based on lexicographic or Hamming distance does not provide significant advantage over random ordering of the terms of the same size.

The paper has presented the experimental results for the performance comparison of five different preprocessing approaches of paths/cuts. For path, we may definitely conclude that the cardinality-based preprocessing (cardinality only or its combinations with lexicographic- and/or Hamming distance- orderings) is worth considering with SPTs as it reduces the computer time. For cut-based SPTs, we could not get the results for the preprocessing method proposed in [3]. We conjecture the method [3] is quite involved and its time complexity to preprocess the cuts is magnitudes higher than the ones we have considered in this paper. Considering this fact in view, we derive the cardinality-based ordering schemes to be a best bet even for cuts too.

References

[1] S. Rai and D.P. Agrawal, *Distributed Computing Network Reliability*, IEEE Computer Society Press, 1990. (Tutorial Text)

[2] C.J. Colbourn, *The Combinatorics of Network Reliability*, Oxford University Press, 1987.

[3] J.A. Buzacott, "The Ordering of Terms in Cut-based Recursive Disjoint Products", *IEEE Trans. Reliability*, vol. R-32, No. 5, Dec. 1983, pp. 472-474.

[4] J.M. Wilson, "An Improved Minimizing Algorithm for Sum of Disjoint Products", *IEEE Trans. Reliability*, vol. R-39, No. 1, Apr. 1990, pp. 42-45.

[5] S. Rai and K.K. Aggarwal, "An Efficient Method for Reliability Evaluation of a General Network ", *IEEE Trans. Reliability*, vol. R-27, Aug. 1978, pp. 206-211.

[6] M.O. Locks, "A Minimizing Algorithm for the Sum of Disjoint Products ", *IEEE Trans. Reliability*, vol. R-36, Oct. 1987, pp. 445-453.

[7] S. Soh and S. Rai, "CAREL: Computer Aided Reliability evaluator for distributed computing networks", *IEEE Trans. Parallel and Distributed Systems*, Accepted.

[8] David Y. Koo, "D/Boolean Application in Reliability Analysis ", *Proceedings Annual Reliability and Maintainability Symposium*, 1990, pp. 294-301.

[9] S. Rai and J. Trahan, "Computing Network Reliability of Redundant MINs", *Proceedings the 21st Southeastern Symposium on System Theory*, Tallahassee, March 1989, pp. 221-225.

[10] S. Hariri and C.S. Raghavendra, "SYREL: A Symbolic Reliability Algorithm Based on Path Cutset Methods", *IEEE Trans. Computers*, C-36(10), Oct. 1987, pp. 1224- 1232.

[11] A. Gmarov, L. Kleinrock and M. Gerla, "A New Algorithm for Network Reliability Computation", *Proceedings of Computer Networking Symposium*, Dec. 1979, pp. 17-20.

[12] A.V. Aho, J.E. Hoffcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, MA: Addison Wesley, 1974.

[13] R.G. Bennetts, "Analysis of reliability block diagrams by Boolean techniques", *IEEE Trans. Reliability*, vol. R-31, June 1982, pp. 159-166.

[14] A. Rosenthal, "Approaches to Comparing Cut-Set Enumeration Algorithms ", *IEEE Trans. Reliability*, vol. R-28, Apr. 1979, pp. 62-65.

[15] J.A. Abraham, "An Improved Algorithm for Network Reliability ", *IEEE Trans. Reliability*, vol. R-28, Apr. 1979, pp. 58-61.

[16] L. Fratta and U.G. Montanari, "A Recursive Method Based on Case Analysis for Computing Network Terminal Reliability ", *IEEE Trans. Communications*, Aug. 1978, pp. 1166-1177.

[17] R.K. Tiwari and M. Verma, "An Algebraic Technique for Reliability Evaluation ", *IEEE Trans. Reliability*, vol. R-29, Oct. 1980, pp. 311-313.

[18] E.J. McCluskey, *Logic Design Principles with emphasis on Testable Semicustom Circuits*, Englewood Cliff, NJ : Prentice Hall, 1986.

Table 4.
Number of Disjoint Paths and CPU Time

BN	R		H		L		C		C and H		C and L	
	DPath	Time	DPath	Time	DPath	Time	DPath	Time	DPath	Time	DPath	Time
N_4^4	5	0	5	0	5	0	4	0	4	0	4	0
N_7^7	12	0	14	0	12	0	7	0	9	0	9	0
N_9^8	16	0	22	0.1	18	0	11	0	11	0	11	0
N_{13}^9	23	0	22	0	26	0.1	16	0	16	0	17	0
N_{13}^{28}	25	0	32	0	26	0.1	15	0	18	0	16	0
N_{14}^{18}	45	0.1	76	0.1	53	0.1	23	0.1	26	0.1	23	0.1
N_{18}^{110}	142	0.1	206	0.1	238	0.1	94	0.1	117	0.1	91	0.1
N_{18}^{36}	59	0.1	58	0.1	55	0.1	30	0.1	25	0	25	0
N_{24}^{19}	73	0.1	101	0.1	76	0.1	39	0.1	41	0.1	41	0.1
N_{20}^{40}	65	0.1	101	0.1	71	0.1	43	0.1	40	0.1	41	0.1
N_{25}^{30}	73	0.1	145	0.1	77	0.1	50	0.1	55	0.1	52	0.1
N_{29}^{29}	112	0.1	157	0.1	126	0.1	76	0.1	82	0.1	78	0.1
N_{42}^{296}	1112	0.4	629	0.3	1098	0.3	502	0.2	459	0.2	579	0.2
N_{44}^{528}	110	0.1	147	0.1	140	0.1	105	0.1	108	0.1	115	0.1
N_{44}^{25}	137	0.1	143	0.1	141	0.1	87	0.1	81	0.1	84	0.1
N_{64}^{78}	789	0.2	1175	0.2	692	0.2	309	0.1	329	0.1	309	0.1
N_{281}^{1300}	6660	0.8	9871	1.3	6621	1.2	2491	0.4	2524	0.5	2540	0.5
N_{281}^{214}	3785	0.6	12311	2.0	7066	1.0	2386	0.6	2230	0.5	2368	0.4
N_{760}^{7776}	159209	76.9	1406735	555.8	174925	95.0	54079	2.1	54428	3.3	53298	2.6

Table 5.
Number of Disjoint Cuts and CPU Time

BN	R		H		L		C		C and H		C and L	
	DCut	Time	DCut	Time	DCut	Time	DCut	Time	DCut	Time	DCut	Time
N_4^4	6	0	5	0	5	0	4	0	4	0	4	0
N_7^7	21	0.1	21	0	16	0	11	0	13	0	12	0
N_9^8	14	0	13	0	13	0	10	0	10	0	10	0
N_{13}^9	27	0	30	0	21	0	14	0	14	0	14	0
N_{13}^{28}	85	0.1	150	0.1	82	0.1	50	0	57	0	50	0
N_{14}^{18}	36	0	26	0	25	0	22	0	25	0	23	0
N_{18}^{110}	364	0.2	740	0.2	315	0.1	155	0.1	134	0.1	150	0.1
N_{18}^{36}	135	0.1	104	0.1	120	0.1	39	0	40	0	38	0
N_{24}^{19}	71	0	124	0.1	76	0.1	40	0	43	0	42	0
N_{20}^{40}	81	0	101	0	78	0	40	0	43	0	43	0
N_{25}^{30}	85	0	125	0.1	62	0	39	0	41	0	43	0
N_{29}^{29}	182	0.1	274	0.1	115	0.1	77	0.1	79	0.1	73	0.1
N_{42}^{296}	2416	1.6	7455	2.9	4544	2.0	1543	1.5	1424	1.6	1386	1.4
N_{44}^{528}	9137	2.7	10753	2.1	11443	1.6	2609	0.4	2852	0.3	3854	0.3
N_{44}^{25}	235	0.1	268	0.1	157	0	79	0	81	0	81	0
N_{64}^{78}	897	0.1	951	0.1	321	0.1	231	0	258	0.1	202	0.1
N_{281}^{1300}	166799	273.8	202563	185.3	61651	63.5	16508	3.0	15924	2.9	16194	3.8
N_{281}^{214}	20881	2.1	21713	3.2	11198	2.4	2339	0.2	2289	0.2	2319	0.2
N_{760}^{7776}	*	**	*	**	1126719	2523.4	317978	184.8	292613	203.4	281453	189.3

BN = Benchmark Network; R = Random ordering; H = Hamming distance ordering; L = Lexicographic ordering;
C = Cardinality ordering; Time is in CPU seconds; * more than 5,000,000 terms; ** more than 10,000 CPU seconds

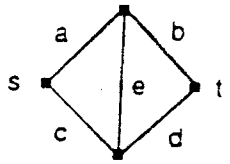


Figure 1

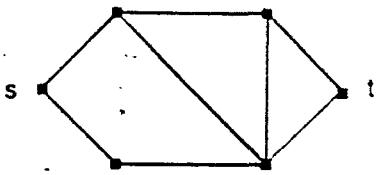


Figure 2

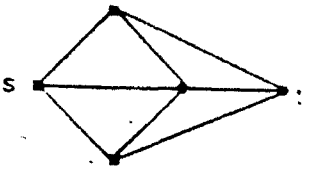


Figure 3

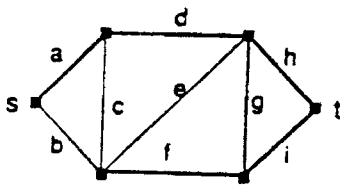


Figure 4

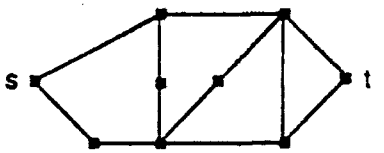


Figure 5

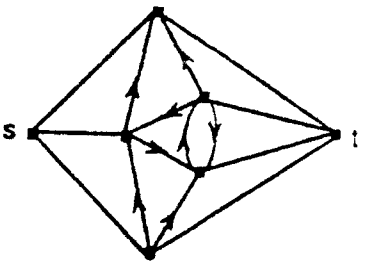


Figure 6

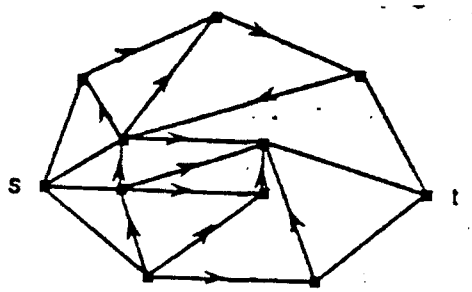


Figure 7

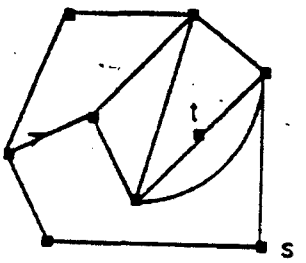


Figure 8

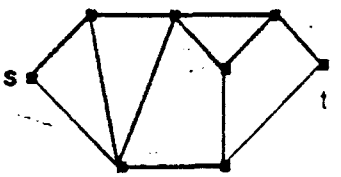


Figure 9

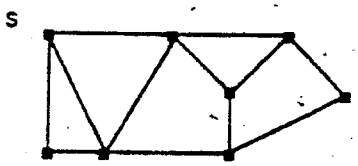


Figure 10

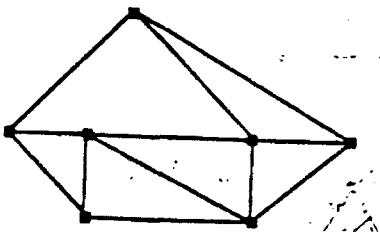


Figure 11

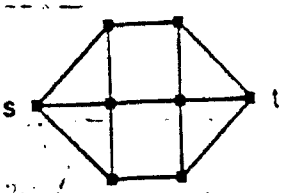


Figure 12

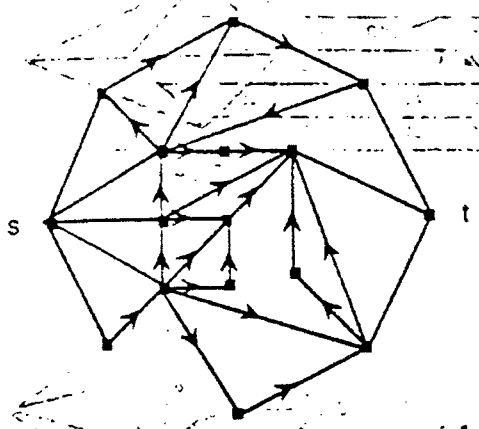


Figure 13

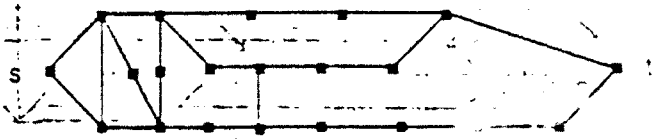


Figure 14

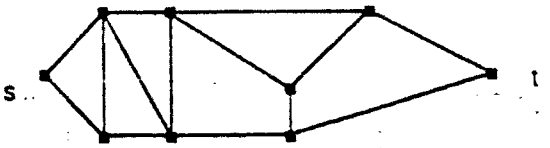


Figure 15

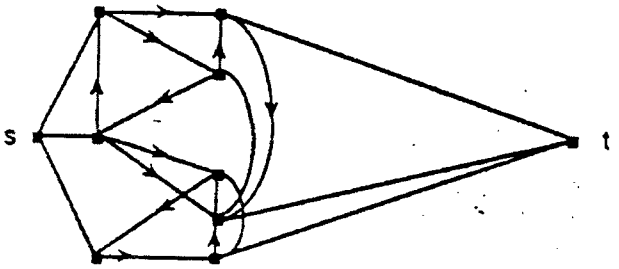


Figure 16

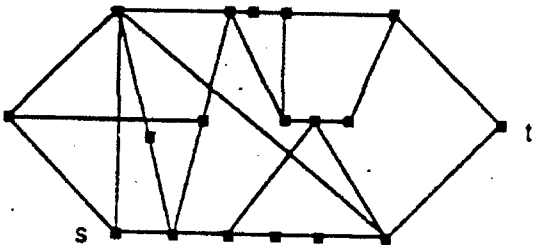


Figure 17

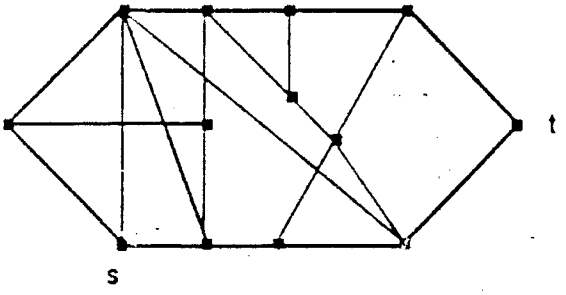


Figure 18

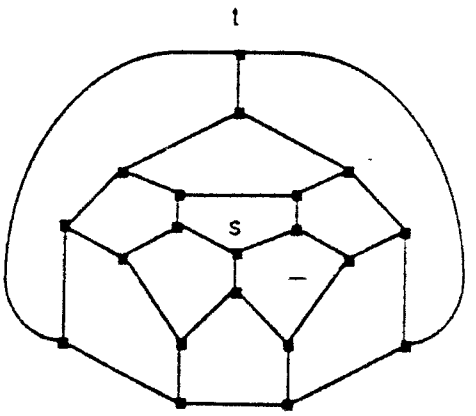


Figure 19

Air Force Office of Scientific Research grant AFOSR-91-0025
Grant duration: October 1, 1990 to October 31, 1992
Report covering the period: October 1, 1991 to October 31, 1992

Report to the Air Force Office of Scientific Research

Multiprocessing Systems: Reliability Modelling and Analysis Using Multimode Components and Dependent Failures

Suresh Rai and Jerry L. Trahan

December 1992

Technical report EE 92-12-001
Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA 70803

Telephone and e-mail:
Suresh Rai — (504) 388-4832, suresh@max.ee.lsu.edu
Jerry L. Trahan — (504) 388-4830, trahan@max.ee.lsu.edu

1. Introduction

To achieve the faster computing speeds imperative for many computer applications, the use of multiple processors operating in parallel is a necessity. Consequently, the reliability of the network interconnecting these processors is of notable importance. For general networks, one may define several reliability measures. *Terminal reliability* is the probability that a working path exists from one specified node to another specified node; *K-terminal reliability* is the probability that a working path exists from one specified node to each of a specified set K of nodes; *broadcast reliability* is a special case of K-terminal reliability in which K is the set of all output nodes. For general networks, the problems of terminal, broadcast, and K-terminal reliability evaluations are computationally intractable, specifically, *#P*-complete. For some particular networks, though, the network offers sufficient structure to allow efficient evaluation of reliability.

Multistage interconnection networks (MINs) are a widely studied means of interconnecting processors to memory or processors to processors by stages of switches. Experimental systems also increasingly use MINs. Such large scale projects as IBM's RP3, the University of Illinois' Cedar, Purdue University's PASM, and PUMPS possess MINs as an integral part of their design. A MIN consists of N inputs and N outputs and (typically) n stages of switching elements (SEs), where $n = \log_2 N$. The most common type of MIN uses as SEs 2×2 crossbar switches, which are able to produce either a straight (T-mode) or exchange (X-mode) connection.

Because of appealing properties such as node and edge symmetry, logarithmic diameter, high fault resistance, scalability, and the ability to host popular interconnection networks, such as ring, torus, tree, and linear array, hypercube multiprocessors have been the focus of many researchers over the past few years. This topology has resulted in several experimental and commercial products, typical examples being the Intel iPSC, NCUBE/10, Caltech/JPL, and the Connection Machine. Conceptually, the hypercube interconnection network is a multidimensional binary cube with a processor at each of its vertices. An n-dimensional hypercube comprises 2^n processors and $n2^{n-1}$ links. Each processor has its own local memory, and processors

communicate by explicit message passing either directly or through some intermediate processors. A subcube is a substructure of a hypercube that preserves the properties of the hypercube.

In the work supported by Air Force Office of Scientific Research grant AFOSR-91-0025, we have examined reliability evaluation problems for MINs and hypercubes. The main aims of the work have been twofold—first, to obtain efficient reliability evaluation algorithms for MINs and hypercubes, and second, to incorporate dependent and multimode failures into these algorithms. The method underlying our research has been to employ a structural approach for studying the reliability of multiprocessing systems based on multistage interconnection networks and hypercubes. We selected this approach for two reasons. First, the regular structure of the topologies studied allows efficient solutions to certain difficult reliability measures. Second, the approach yields a framework in which incorporating dependent and multimode failures is straightforward. For our work this second year of the project, we concentrated on new computational procedures for calculating reliability measures (exactly and approximately) in MIN and hypercube based multiprocessing architectures. Sections 2 and 3 provide the results. A bibliography at the end of this report compiles our research papers on the topic. Appendix A summarizes results obtained in the first year (October 1, 1990 to September 30, 1991). We believe that our efforts will help the AFOSR in the areas of analysis, modelling and simulation of these multiprocessor systems connectivity, survivability and availability (with or without multimode component and dependent failure models).

In addition to the efforts of the P.I., S. Rai, and co-P.I., J. Trahan, the work was supported by a doctoral student, S. T. Soh, and five M.S. students: S. Ananthakrishnan, V. Narayan, T. Smailus, N. Venkataramani, and D. Wang.

2. Multistage Interconnection Networks

2.1. Results - An Overview

Our aims in studying MINs were to develop efficient algorithms for the evaluation of terminal reliability (TR), broadcast reliability (BR), and K-terminal reliability (KR) of MINs.

These algorithms were to incorporate dependent and multimode failures. In the first year of the project, we developed algorithms for TR, BR, and KR of the shuffle-exchange network with an extra stage (SENE) [2, 9]. These algorithms were efficient, each running within $O(\log N)$ time for an $N \times N$ SENE. They assumed two-mode and independent failures, however. This work was an essential first step in the development of more general algorithms, as the simplifying assumptions of two-mode and independent failures allowed us to concentrate on exploiting the structural properties of the MIN to develop reliability evaluation algorithms. We expected, and later results have borne out the expectation, that these structure-based algorithms would provide the basis for algorithms under more realistic assumptions. Later in the first year, we extended these algorithms to incorporate multimode switch failures [18]. The new algorithms ran to within a constant factor of the time of the algorithms assuming two-mode failures, with the exception of the TR algorithm, which ran in $O(\log N)$ time as opposed to $O(\text{loglog } N)$ time.

In the second year of the project, we enhanced the algorithms to incorporate dependent failures [3, 6, 16]. We also examined the SENE and the merged delta network (MDN) under conditions that allowed link failures [10, 17]. In what follows, we discuss the results in brief; the details can be obtained from the enclosed papers.

2.2. Multimode and Dependent Failures

Assumptions of two-mode and independent failures are common in reliability evaluation, since they simplify the computation and since the evaluation problems remain intractable for general networks even under these assumptions. These assumptions, however, fail to adequately model real-world situations. For example, researchers have described instances of dependent failures, or fault side-effects, in the PASM. Moreover, the assumption of independent failures leads to an overestimate of reliability. The two-mode model leads to an underestimate of reliability because it does not allow a degraded operational mode of SEs. Note, some work exists on incorporating multimode components or dependent failures into reliability analysis for telecommunication networks, though little for specific multiprocessor networks.

We developed efficient algorithms for TR, BR, and KR evaluation of an SENE composed of identical SEs, allowing multimode and dependent failures. The SENE that we examined contains 2×2 SEs. In the two-mode case, an SE is either completely working or completely failed. In the multimode case, the algorithms assume a 4-mode model of an SE: a fully operational mode, a completely failed mode, and two degraded operational modes, namely, stuck-at T mode and stuck-at X mode. There exist 16 possible degraded modes for a 2×2 crossbar SE. One can readily extend our approaches to incorporate any more or all of the 16 modes. We model stuck-at faults (0 or 1) and bridging faults between two adjacent links in terms of switch failures. To incorporate multimode failures, thereby extending the earlier reliability evaluation algorithms [2, 9] to more realistic assumptions, we modified the shock model of dependent failures, which is considered in the literature for general networks. For an $N \times N$ SENE, the algorithms run in time $O(\log N)$, $O(\log N)$, and $O(k \log N)$, respectively, where $k = |K|$.

The shock model assumes that statistically independent *shocks*, which occur with known probability, cause the failure of network components. When a shock occurs, it causes the failure of a specific component or set of components. We say that the shock *affects* the component or set of components. A shock affecting a single component is called an *internal shock* (IS), while a shock affecting multiple components is called an *external shock* (ES). The main motivation for the shock model is that there exist events that may cause one or more components (links or nodes) in a network to fail simultaneously. Explicitly, for example, some components may share important equipment in common such as a power supply, or several components may reside in a common chip. Components will fail simultaneously if power fails or the chip burns. Test-maintenance-operation errors, design defects, and electromagnetic interference may also lead to simultaneous failures of components. Some shocks may not induce failures of components but may degrade them so that their joint probability of simultaneous failure increases.

The shock model was defined for two-mode components. By expanding the notion of an IS, we generalize its application to an SENE in which SEs may operate in degraded modes. Instead of associating a single IS with a single component, we associated one IS with each failed

or degraded working mode of a component. We model a shock that causes a stuck-at fault in a control line by a shock that affects the SE to be stuck-at T or X mode.

For external shocks, we restrict our analysis to two modes only, so the occurrence of an ES causes all affected SEs to fail. SEs that are far apart are unlikely to be affected by a single ES; SEs that are adjacent, however, are likely to be affected by one shock. In general, the classes of ESs that we define are motivated by the failure of one SE causing the failure of other SEs due to the links connecting them. For shared-memory computers, each processor reads from or writes to the shared-memory through a MIN, so communication flows in both directions. When the forward (reverse) part of an SE is failed, this SE may send erroneous routing and control information to either or both SEs in the next (previous) stage that are connected to it and may cause one or both to fail. Hence, we assume that an external shock causes a failure in adjacent SEs either in the forward direction (toward the network outputs) or in the reverse direction (toward the network inputs). In a practical design, a failure of an output (input) controller can create a forward (reverse) external shock. Researchers have described such dependent failures in the PASM.

To help compute the reliabilities, we define four *classes* of shocks. Each class of shock will affect a certain structured set of SEs. A Class 1 shock will affect an SE to be completely failed, stuck-at T mode, or stuck-at X mode. A Class 2 shock will affect an SE and one SE to which it is connected in the previous or next stage to be failed. A Class 3out shock will affect an SE and the two SEs in the next stage to which it is connected by its output links to be failed. A Class 3in shock will affect an SE and the two SEs in the previous stage to which it is connected by its input links to be failed. We define such shocks for every SE. Because the structures affected by the same class of shocks are the same and all SEs are identical, the probabilities that the same class of shock occurs are identical for all such shocks. The purpose of our work is, then, to find the relationship between the probabilities of each class of shock and the reliability of the whole network.

In earlier work [2, 9], we noted that SENE paths form a simple series-parallel graph for TR and a pair of intersecting binary trees for BR and KR. To incorporate dependent and multimode

failures, we again followed this concept, but also were required to include a careful and detailed accounting of the shocks that may affect the SEs on the paths. Note that a single shock may affect one, two, or three SEs on the relevant paths.

As an example, consider the computation of K-terminal reliability in an $N \times N$ SENE. We outline it simply as follows. Let E be a Boolean variable such that $E = 1$ if and only if working paths exist to all outputs in set K from input s. Then $P(E)$ is the K-terminal reliability. Let I_f, I_t, I_x, and I_w denote the events that SE I is failed, stuck-at T, stuck-at X, and working, respectively. Using the theorem of total probability and noting that $P(E \mid I_f) = 0$, we have

$$KR(K,N,s) = P(E|I_t)P(I_t) + P(E|I_x)P(I_x) + P(E|I_w)P(I_w). \tag{1}$$

Equation (1) helps us obtain the K-terminal reliability by computing each term using the following steps.

Step 1. Determine individual probabilities $P(I_t)$, $P(I_x)$, and $P(I_w)$.

Step 2. Compute conditional probability terms $P(E \mid I_t)$, $P(E \mid I_x)$, and $P(E \mid I_w)$.

Step 3. Obtain $KR(K,N,s)$ using the results of Steps 1 and 2.

Steps 1 and 3 of this procedure are straightforward. Step 2 requires a careful case analysis dependent on the degraded working modes of the SEs and on the composition of the set K. For K-terminal reliability, we describe each SE on a path from a specified input s to an output in K as *marked*.

For an input s and a set K of outputs in an $N \times N$ SENE, where $k = |K|$, the equation below computes the K-terminal reliability in $O(k \log N)$ time. Let p_f, p_t, p_x, p_2, p_{3o}, and p_{3i} denote the probabilities of a Class 1 shock that causes failure of an SE, a Class 1 shock that causes an SE to be stuck-at T, a Class 1 shock that causes an SE to be stuck-at X, a Class 2 shock, a Class 3out shock, and a Class 3in shock. Let $p_w = 1-(p_t + p_x + p_f)$, $q_2 = 1-p_2$, $q_{3o} = 1-p_{3o}$, and $q_{3i} = 1-p_{3i}$.

$$KR(K,N,s) = \left[(p_t + p_x - 2p_w p_{3o} q_{3o}^{-1})KR_1(K,N,s) + p_w q_{3o}^{-1} KR_2(K,N,s)\right] q_2^2 q_{3o} q_{3i}^2.$$

Recurrence expressions compute values for $KR_1(K,N,s)$ and $KR_2(K,N,s)$ according to the different cases enumerated below. Computing the base case for $N = 4$ requires consideration of several different cases. For the sake of brevity, we do not enumerate the results for the base case.

Case I: Only one child of considered SEs is marked.

1. If the T mode in these SEs allows working paths from input s to the k outputs, then

   $KR_1(K,N,s)$ and $KR_2(K,N,s)$ can be computed by

$$KR_1(K,N,s) = (p_t + p_w)q_2^3(q_{3o}q_{3i})^2 KR_1\left(K,\frac{N}{2},s\right)$$

$$KR_2(K,N,s) = \left[(p_t + p_w)^2 KR_2\left(K,\frac{N}{2},s\right) + 2(p_t + p_w)(p_x + p_f + \alpha^{-1}p_w - 1)KR_1\left(K,\frac{N}{2},s\right)\right] \cdot q_2^6(q_{3o}q_{3i})^4.$$

2. If the X mode in these SEs allows working paths from input s to the k outputs, then

   $KR_1(K,N,s)$ and $KR_2(K,N,s)$ can be computed by the equations above, exchanging p_t and p_x.

Case II: Both children of considered SEs are marked. Let K_l and K_r be the subsets of K that lie in the left and right subgraphs, respectively.

$KR_1(K,N,s)$ can be computed by

$$KR_1(K,N,s) = KR_1\left(K_l,\frac{N}{2},s\right)KR_1\left(K_r,\frac{N}{2},s\right) \cdot p_w q_2^3 q_{3o} q_{3i}^2.$$

$KR_2(K,N,s)$ can be computed by

1. If input s can access the right (left) subgraph when the considered SEs are set in T (X) mode, then

$$KR_2(K,N,s) = \Bigg[p_w^2 q_{3o}^{-2} KR_2\left(K_l,\frac{N}{2},s\right) KR_2\left(K_r,\frac{N}{2},s\right)$$
$$+ 2p_l p_w q_{3o}^{-1} KR_1\left(K_l,\frac{N}{2},s\right) KR_2\left(K_r,\frac{N}{2},s\right)$$
$$+ 2p_x p_w q_{3o}^{-1} KR_2\left(K_l,\frac{N}{2},s\right) KR_1\left(K_r,\frac{N}{2},s\right)$$
$$+ 2\left(p_w\left(p_f + \alpha^{-1}p_w - 1\right)q_{3o}^{-1} + p_l p_x\right) KR_1\left(K_l,\frac{N}{2},s\right) KR_1\left(K_r,\frac{N}{2},s\right)\Bigg]$$
$$\cdot q_2^6 q_{3o}^4 q_{3i}^4.$$

2. If input s can access the left (right) subgraph when the considered SEs are set in T (X) mode, then KR_2 can be computed by the equation above, exchanging p_l and p_x.

In summary, we used the shock model to develop efficient algorithms for terminal, broadcast, and K-terminal reliability evaluation of an SENE with both dependent and multimode failures. All previous work in this area assumed that failures of SEs in MINs are independent and that each component has only two possible modes. In the real world, however, these assumptions are not often true. Our algorithms for broadcast and K-terminal reliability evaluation run within a constant factor of time of the algorithms that we developed [2, 9] for the same problems under assumptions of independent and two-mode failures. Our algorithm for terminal reliability runs in $O(\log N)$ time as opposed to $O(\text{loglog } N)$ time for the earlier TR algorithm that assumed independent and two-mode failures. Though we developed the algorithms for the SENE MIN, the principles underlying the incorporation of dependent failures into reliability evaluation algorithms readily generalize to apply to reliability evaluation algorithms for any other regularly structured MIN, such as those reported in the literature for the generalized INDRA network, merged delta network, and augmented C network. Similarly, though we developed the algorithms for an SE model that includes only 4 of the 16 possible modes of operation, degraded operation, and failure, the underlying principles readily generalize to incorporate more or all of the possible modes by simply including the appropriate cases. Finally, though we developed the algorithms for a

particular set of dependences between failures of SEs, one can use the shock model in the same fashion to incorporate other dependences between failures.

2.3. LINK FAILURES

Most studies carried out to evaluate the reliability measures of a MIN have assumed that only nodes are susceptible to failure or accounted for the failure of links by considering a link as part of the adjacent switching element. Neither approach accurately accounts for link failures, even though the failure of even a single link disconnects several input/output paths, leading to a lack of fault tolerance and low reliability. We have designed simple and efficient algorithms for the TR and BR evaluation of the SENE and the merged delta network (MDN) under the assumption that both nodes and links can fail [10, 17]. We assumed two-mode and independent failures. This extension to consider link failures is in line with our other work on incorporating more realistic assumptions into the reliability evaluation of the regularly structured networks in MINs and hypercubes.

For the SENE, the TR and BR evaluation algorithms run in $O(\log\log N)$ and $O(\log N)$ time respectively. For the MDN, the algorithms each run in $O(\log N)$ time. These times are within a constant factor of the times required for evaluation of the corresponding expressions that we developed [2, 9] for the SENE and the ones reported for the MDN, though these earlier expressions assumed that only SEs may fail and links are always working. Hence we can conclude that incorporating both node and link failures into the reliability analysis for the SENE and MDN does not add any extra overhead to the computation time.

3. Hypercubes

3.1. RESULTS - AN OVERVIEW

In the first year of the project, we explored the hypercube reliability problem with deterministic and probabilistic models. A deterministic model assumes a given set of failures in a cube. Specifically, the problem was to determine the size and location of the maximal dimension

available (fault-free) subcube. We denote this as the reconfiguration problem. We developed a set of algorithms, each based on a different representation of a hypercube, to solve the reconfiguration problem. Additionally, we extended the concepts developed in the reconfiguration algorithms to address the problem of dynamic allocation of subcubes of a hypercube to multiple tasks.

Probabilistic fault tolerance measures for hypercube multiprocessors are useful for packet-switching applications because they verify the sturdiness of the topology and depict the probability of successful flooding (for route set up or packet transmission). To study the probabilistic model, we looked into terminal and network reliability evaluations using CAREL, a tool used to compute general network survivability measures. (The Year 1 report in Appendix A describes CAREL.) Because of the exponentially large number of paths (spanning trees) involved as a starting step for solving the terminal (network) reliability problem, even the efficient general algorithms in CAREL failed to generate results for hypercubes of dimension $n \geq 4$ in reasonable time [13].

In the second year of the project, we continued our efforts to discover efficient approaches to solve reliability problems in a hypercube architecture. For the deterministic model, we solved the following specific problems:

(a) In a multiuser-multitasking environment, subcube allocation plays an important role. We obtained an efficient distributed algorithm for largest operational subcube identification.

(b) The K-Connected Functionality (KCF) problem applies to large scale degradable hypercubes used to run concurrent algorithms that are not sensitive to changes in the system topology. We established a bound on the number of faulty nodes that an n-dimensional hypercube, C_n, can tolerate such that at least K processors remain connected, provided there are no C_0 or C_1 disconnections.

(c) We constructed a fault tolerant broadcasting algorithm useful for distributed agreement and clock synchronization.

Section 3.2 discusses this work in greater detail.

In addition to broadening our exploration towards deterministic models, we developed methods in the probabilistic model for approximating network and terminal reliabilities using lower

and upper bounds, between which the exact measure is guaranteed to exist. Section 3.3 describes this work.

3.2. DETERMINISTIC MODEL

In the first year of the project, we obtained a centralized algorithm for reconfiguration of a hypercube after faults [5]. In the second year of the project, we improved on this algorithm [1].

A related and equally important issue is that of largest operational subcube (LOS) identification. This year, we proposed a distributed LOS algorithm [13]. A distributed algorithm allows each PE v to run a program to obtain a subcube with maximum size k that contains v. Our method uses the CMB operator of CAREL which includes the multiple variable inversion concept. The computational complexity of the CMB operator is data dependent. Hence, we consider the best and worst cases of the LOS approach. We proved that LOS takes $O(m^2)$ steps for the best case while $O(m^4)$ for the worst case, where $m \leq n$ denotes the number of faulty nodes in a hypercube C_n. A heuristic, based on a modification of the LOS approach, however, generates correct results in time $O(m^3)$ for 99% of the problem instances tested. In case the number of non-available nodes (faulty or busy) increases, an alternative distributed approach processes w available nodes in $O(wn)$ time to solve the LOS problem [13].

The KCF model uses the concept of forbidden faults and considers as forbidden fault sets that cause disconnection of a working 0-subcube, C_0. Researchers have shown that a C_n can tolerate up to $2n-3$ faulty nodes and remain connected provided that the failures do not disconnect any C_0 subcube. Our work further generalizes the KCF connectedness measure by extending the forbidden set to include C_1 disconnections. We established that a C_n can tolerate up to $\min\{3n-5, 4n-9\}$ faulty nodes and remain connected if disconnections of C_1 or its subsets do not occur. This assumption is not impractical as researchers have studied the probabilities of C_i disconnection and have shown that for C_1 it is very low.

We adopted a hybrid approach to fault-tolerant broadcasting that uses the concepts of redundant and non-redundant methods. It, thus, avoids faulty PEs in the communication paths to

improve the capability of an existing redundant type algorithm. Here, each PE sends the message only to its healthy neighbors. Furthermore, the algorithm modifies the message reception mechanism to recognize only the first arriving copy of the message and to ignore later redundant copies [13].

3.3. PROBABILISTIC MODEL

Our work on the probabilistic model concentrated mainly towards generating lower bounds for network and terminal reliabilities. A lower bound on reliability is quite appealing because it can be obtained with substantially less computation than an exact bound, and the system will be at least as reliable as the bound. Towards obtaining bounds on network reliability, we first solved Spemer bounds and Kruskal-Katona bounds for hypercubes. These solutions, however, required unreasonable computation time for cubes of dimension greater than 3. We next attempted exact reliability evaluation using spanning trees and CAREL. An n-cube is a matroid and CAREL solves reliability problems in time polynomial in the number of spanning trees, $ST(C_n)$. Unfortunately, $ST(C_n)$ is exponential with respect to n. Thus, this technique is not advisable for large n. As an alternative to these solution approaches, we derived a tighter lower bound on NR using structural properties of the hypercube [4, 8, 13]. The NR bounding algorithm uses the structure of a C_n to recursively generate a lower bound from the knowledge of the lower bound for a C_{n-1}. One can partition a C_n into two C_{n-1}'s. We define an *exterior link* as a link between these two C_{n-1}'s. The algorithm divides the problem into three mutually disjoint cases such that events in each case are also mutually disjoint among themselves. Thus, the lower bound on NR is the sum of the lower bounds on NR obtained from the following three cases.

Case 1. Both $(n-1)$-cubes and only one exterior link operate.

Case 2. All 2^{n-1} exterior links operate.

Case 3. For $2 \le i \le 2^{n-1}-1$, i exterior links and one $(n-1)$ cube operate.

Case 3 is further subdivided for $i = 2^{n-1}-1$ and $i = 2^{n-1}-2$ to help solve large C_n problems efficiently. Note, the contribution to NR from the Case 3 terms where $2 \le i \le 2^{n-1}-3$ is very small and so is computed by one expression for all such terms.

The terminal reliability bounding algorithm for a source s and terminal t at distance n from each other considers only shortest paths from s to t [4, 8, 13]. (If we consider only shortest paths, then, without loss of generality, we may consider these as nodes 0 and 2^n-1 in a C_n.) Of the $n!$ (s,t) paths, n of these are node and link disjoint. We efficiently computed a lower bound on TR from a specific set of $\alpha \cdot n$ paths, where α represents a multiplier. A routing algorithm determines the parameter α. We show that:

(a) $\alpha = \lceil (n/2)-1 \rceil$ if the following properties are satisfied.

Property N1: Paths $P_{i,j}$ and $P_{i,l}$ are node and link disjoint, for $j \neq l$.

Property N2: Paths $P_{i,j}$ and $P_{k,j}$ have $(i+1)$ common nodes and links for $i < k$.

Property N3: Paths $P_{i,j}$ and $P_{k,l}$ are node and link disjoint for $j \neq l$.

(b) $\alpha = (n-2)$ if the following properties are satisfied.

Property L1: Paths $P_{i,j}$ and $P_{i,l}$ are link disjoint for $j \neq l$.

Property L2: Paths $P_{i,j}$ and $P_{k,j}$ have $(i+1)$ common links, for $i < k$.

Property L3: Paths $P_{i,j}$ and $P_{k,l}$ are link disjoint, for $j \neq l$.

In (a) and (b), let $1 \le i, k \le \alpha$ and $1 \le j, l \le n$. The algorithm exploits properties N1 through N3 and L1 through L3 to generate TR bounds for node, link, and node and link failure cases. We proved that our TR bound is tighter than previously established bounds based on disjoint shortest paths. To improve the bounds, we have also constructed a 2-cube model for bounding TR of a C_n. A combination of the Boolean and 2-cube approaches leads to a still tighter bound on TR.

Note that the fault model above allows both node and link failures. The inclusion of link failures implicitly considers multimode failures of processors as follows. A processor connects to a link through an I/O port and associated control circuitry. For example, the functional block of a node in the Intel iPSC architecture comprises 64K bytes EPROM, 512K bytes DRAM, an Intel 80286 processor, and an Intel Ethernet Controller 82586 for I/O. The failure of a port or its

controller is actually a partial failure of a processor, resulting in a degraded operational mode for the processor. We model this multimode failure of a processor as a two-mode failure of a link.

4. General Results

In addition, we computed the node and link failure probabilities in a typical hypercube system [13]. To illustrate this, assume an Intel iPSC architecture. Based on the functional block described above for its node, we utilized MIL-HDBK-217F to determine relevant parameters for the chips and computed mean-time-to-failure rates. A link, in this case, includes an I/O unit at a node, physical communication media, and the I/O unit of the adjacent node. We have also expanded our effort towards the understanding of an object-oriented approach to evaluate fault trees with dynamic and non-dynamic gates. HARP, a typical fault tree solver, uses these gate types to model the behavior of a fault tolerant hypercube. We used an object-oriented approach to solve fault trees directly [12,15]. This effort is different from the one given in HARP where an indirect approach is utilized and the fault tree is inherently translated into a Markov model. All these efforts are helpful in understanding different aspects of the hypercube reliability evaluation problem.

5. Future Work

Regarding MINs, the most significant open problem related to our work is to obtain efficient algorithms for evaluating the Network Reliability (NR) of MINs. We expect that efficient algorithms exist because of the regular structure of MINs and because of results on other reliability problems. Following the principles we have established, an efficient algorithm for NR under assumptions of independent failures and two mode components should be able to be readily generalized to an algorithm under assumptions of dependent failures and multimode components.

Regarding hypercubes, the most significant open problem related to our work is to obtain reliability evaluation algorithms that incorporate dependence between failures. As for MINs, we

believe that the shock model can provide a basis for incorporating dependent failures into reliability evaluation algorithms.

Bibliography

PAPERS PUBLISHED OR ACCEPTED FOR PUBLICATION

[1] S. Rai and J. L. Trahan (1992), "A Reconfiguration Technique for Fault Tolerance in a Hypercube," to appear in *Parallel Processing Letters*.

[2] J. L. Trahan and S. Rai (1992), "Reliability Evaluation of Extra Stage Shuffle-Exchange MINs," to appear in *Proc. 1992 International Conference on Computer Communication* (International Council for Computer Communication, Genova, Italy, Oct. 1992).

[3] J. L. Trahan, D. X. Wang, and S. Rai (1992), "Incorporating Dependent and Multimode Failures into Reliability Evaluation of Extra Stage Shuffle-Exchange MINs," to appear in *Proc. 30th Allerton Conf. on Communication, Control, and Computing* (Univ. of Illinois, Monticello, IL, Oct 1992).

[4] S. T. Soh, S. Rai, and J. L. Trahan (1992), "Improved Lower Bounds on the Reliability of Hypercube Architectures," *Proc. 5th ISMM Conf. on Par. and Distr. Computing and Systems* (International Soc. Mini and Micro Computers, Pittsburgh, PA, Oct. 1992), pp. 182-187.

[5] S. Rai and J. L. Trahan, "ATARIC: An Algebraic Technique to Analyse Reconfiguration for Fault Tolerance in a Hypercube," *Proc. 3rd IEEE Symp. Par. and Distr. Processing* (IEEE Computer Society, Dallas, TX, Dec. 1991), pp. 548-555.

PAPERS SUBMITTED

[6] J. L. Trahan, D. X. Wang, and S. Rai (1992), "Incorporating Dependent and Multimode Failures into Reliability Evaluation of Extra Stage Shuffle-Exchange MINs," submitted to *IEEE Trans. Par. and Distr. Sys.*

[7] S. Rai, J. L. Trahan, and T. Smailus (1992), "Processor Allocation in Faulty Hypercube Multiprocessors," submitted to *IEEE Trans. Par. and Distr. Sys.*

[8] S. T. Soh, S. Rai, and J. L. Trahan (1992), "Improved Lower Bounds on the Reliability of Hypercube Architectures," submitted to *IEEE Trans. Par. and Distr. Sys.*

[9] J. L. Trahan and S. Rai (1991), "Reliability Evaluation and Decision Problems in Extra Stage Shuffle-Exchange MINs," submitted to *Networks*, special issue on interconnection networks.

[10] J. L. Trahan and V. K. Narayan (1992), "Reliability Evaluation of Redundant Path Multistage Networks with Node and Link Failures," submitted to 31st ACM Southeast Conference.

[11] S. Rai, J. L. Trahan, and T. Smailus (1992), "Processor Allocation in Faulty Hypercube Multiprocessors," submitted to 1993 IEEE International Symposium on Circuits and Systems.

[12] N. Venkataramani and S. Rai (1992), "An Object Oriented Approach to Evaluate Fault Trees Having Dynamic and Non-Dynamic Logic Gates," submitted to 1993 IEEE International Symposium on Circuits and Systems.

PH.D. DISSERTATION / M.S. THESES

[13] S. Soh (Ph.D., 1993), "Modeling Reliability in Hypercube Architecture."

[14] T. O. Smailus (M.S., 1992), "On Processor Allocation / Deallocation Techniques in Hypercubes."

[15] N. Venkataramani (M.S., 1992), "An Object Oriented Approach to Evaluate Fault Trees Having Dynamic and Non-Dynamic Logic Gates."

[16] D. X. Wang (M.S., 1992), "Incorporating Dependent and Multimode Failures into Reliability Evaluation of Multistage Interconnection Networks."

[17] V. K. Narayan (M.S., 1992), "Reliability Evaluation of Redundant-Path Multistage Networks with Node and Link Failures."

[18] S. C. Ananthakrishnan (M.S., 1992), "Reliability Evaluation of Multistage Interconnection Networks with Multimode Failures."

To appear in Parallel Processing Letters

A RECONFIGURATION TECHNIQUE FOR FAULT TOLERANCE IN A HYPERCUBE

SURESH RAI and JERRY L. TRAHAN

Department of Electrical and Computer Engineering
Louisiana State University, Baton Rouge, LA 70803

ABSTRACT

The hypercube architecture is a popular topology for many parallel processing applications. For continued operation of the hypercube multiprocessors after the failure of one or more 1-subcubes and/or links, fault tolerance by reconfiguration is an important problem. This paper considers the reconfiguration issue and presents an algebraic technique to analyze the problem, extending the concepts in [9]. The technique uses algebraic operators to identify the maximum dimensional fault-free subcube, and thus helps in achieving graceful degradation of the system. We analyze the complexity of our algorithm and show that it is efficient as compared to previous algorithms [1, 8, 14].

Keywords: hypercube reconfiguration, subcube recognition, fault-free subcubes.

1. Introduction

The suitability of a multiprocessing architecture is largely affected by its ability to tolerate faults. After identification of faulty elements, reconfiguration of the multiprocessor and the distributed algorithm running on the multiprocessor allows graceful degradation. Reconfiguration ensures continued operation of a hypercube multiprocessor after the failure of one or more subcubes (a subcube is a subgraph of a hypercube that preserves the properties of the hypercube) and/or links. Algorithms exist for diagnosing faulty processors and links in hypercubes [2, 3]. Fortunately, most parallel algorithms can be formulated with the dimension n of the hypercube being a parameter of the algorithm [1]. Hence, the reconfiguration problem in a hypercube multiprocessor reduces to identifying the maximum dimensional fault-free subcube(s).

Becker and Simon [1] provided a procedure that usually, but not always, finds the maximum dimension "d" of a fault-free subcube. Özgüner and Aykanat [8] utilized the principle of inclusion-exclusion in algorithms that always find d and also the number of fault-free d-subcubes or the complete set of fault-free d-subcubes. Kim *et al.* [5] presented a top-down processor allocation strategy that also applies to the reconfiguration problem. Sridhar and Raghavendra [14] gave an algorithm for reconfiguration based on assigning weights to nodes and identifying subcubes based on their maximum and minimum weight nodes. Latifi [6] gave a distributed algorithm that follows a greedy heuristic for identifying the largest fault-free subcube in a faulty hypercube. Most of these techniques treat a link as a 1-subcube. When the end nodes of a link are not faulty, treating a link as equivalent to a 1-subcube is erroneous.

This paper introduces a new algebraic technique that returns a list of fault-free subcubes for the purpose of reconfiguration after faults in a hypercube. Our technique has

This work was supported in part by the Air Force Office of Scientific Research under grant AFOSR-91-0025.

the advantages of simplicity and improved time complexity over previous exact methods. We present four operators, namely # (sharp), S (dollar), D, and pl to help describe our method. The proposed technique is formulated to run on a single processor which would typically be the host or the resource manager in a commercial hypercube system.

The layout of the paper is as follows. Section 2 discusses the hypercube and its properties and presents a fault model that allows subcube and link failures. The algebraic operators are given in Section 3. Section 4 describes the algorithms and illustrates the technique with examples. The complexity issues presented in Section 5 show that our method is more efficient than previous approaches.

2. Hypercube Concepts and Fault Models

An n-dimensional hypercube is defined as $Q_n = K_2 \times Q_{n-1}$, where K_2 is the complete graph with two nodes, Q_0 is a trivial graph with one node and $\times$ is the product operation on two graphs [4]. Let Q_n be modeled as a graph $G(V, E)$ with $|V| = 2^n$ and $|E| = n\,2^{n-1}$. The graph $G(V, E)$ is both node and link symmetric. Each node in $G(V, E)$ represents a processor and each edge represents a link between a pair of processors. Assign binary numbers from 0 to (2^n-1) to nodes such that addresses of any two adjacent nodes differ in only one bit position. The reader is suggested to refer to [4, 11] for other interesting properties of a hypercube graph.

Using an n-tuple, a processor in Q_n is denoted by $b_{n-1}\ldots b_i\ldots b_0$, where $b_i \in \{0, 1\}$. Two adjacent nodes which differ in the ith bit are said to be in direction i $(0 \le i \le n-1)$ with respect to each other. A subcube in a hypercube Q_n is a subset of a hypercube that preserves the properties of a hypercube. It is represented by an n-tuple $\{0,1,x\}^n$. Coordinate values "0" and "1" can be referred to as fixed or bound coordinates and "x" as free. An i-dimensional cube (or i-subcube, Q_i) in Q_n has $(n-i)$ bound coordinates and i free coordinates. Note that we will use the terms node and 0-subcube interchangeably throughout the paper since they denote the same object. We describe a link with an n-tuple $\{0,1,q\}^n$ containing exactly one q. The position of the coordinate q in one of the n coordinate positions indicates the adjacency direction for the end nodes. For example, 100q denotes the link with end nodes 1000 and 1001 in Q_4. Note that both x and q cannot be present in the n-tuple representation of a Q_i. This notation differentiates a link from a 1-subcube. We refer to the node, subcube, and link notation described above as *ternary vector* (TV) notation.

When an i-subcube is faulty, we assume that all 2^i nodes forming the i-subcube along with their interconnecting links are unavailable. Node failure is a special case of an i-subcube fault, where $i = 0$. We assume that a node failure removes the node and all incident links from the graph. A link failure has the effect of deleting the particular link from $G(V, E)$.

Note that a link and/or node may be faulty due to a hardware failure. When some task is currently being executed on an i-subcube, the said i-subcube is temporarily unavailable and may also be considered as faulty from the viewpoint of reconfiguring the multiprocessor to run an additional task.

3. Algebraic Operators

This section defines the algebraic operators, #, S, D, and pl, that we will use to find maximum dimension non-faulty subcubes in the presence of subcube and link failures. Each operator "o" works on pairs of n-tuples $\{0, 1, x\}^n$ or $\{0, 1, q\}^n$. Each definition begins with a table describing the "o" operation on each pair of corresponding elements from the n-tuples, then uses this coordinatewise definition to define the "o" operation on a pair of n-tuples. In what follows, c_h describes a working subcube, while f_s represents a (failed) subcube or link. Algorithm 1 uses the # and S operators to produce a set of maximal size subcubes contained in c_h and disjoint from f_s. Algorithm 2 operates on faults using the D-operator, and uses the pl operator and equality checking to remove redundant terms from the computation of the fault-free subcubes.

Definition 1: # operator. Let $c_h = a_{n-1} \ldots a_i \ldots a_0$ and $f_s = b_{n-1} \ldots b_i \ldots b_0$, where $a_i \in \{0,1,x\}$ and $b_i \in \{0,1,x\}$, where the fault type is a subcube failure. Table 1 defines the coordinate # operation.

The following equation defines # operation between c_h and f_s.

$$c_h \# f_s = \begin{cases} c_h & ; \text{ if } a_i \# b_i = y \text{ for any } i \\ \varnothing & ; \text{ if } a_i \# b_i = z \text{ for all } i \\ \bigcup_{i \in P} a_{n-1} \ldots a_{i+1} \alpha_i a_{i-1} \ldots a_0 & ; \text{ otherwise, where} \\ & \qquad P = \{ i \mid a_i \# b_i = \alpha_i = 0 \text{ or } 1 \} \end{cases} \tag{1}$$

If $C = \{c_1, \ldots, c_r\}$ is a set of n-tuples, then let $C \# f_s = \bigcup_{h=1}^{r} c_h \# f_s$. Miller [7] described the sharp (#) operator and its properties. The # operator is quite general and a modification to it finds use in PLA testing [10] and reliability computation of general networks [13, 15]. Note that in the coordinate # operation, y denotes that the cubes are disjoint and z indicates a possible overlap.

Definition 2: S operator. Let $c_h = a_{n-1} \ldots a_i \ldots a_0$ and $f_s = b_{n-1} \ldots b_i \ldots b_0$, where $a_i \in \{0,1,x\}$ and $b_i \in \{0,1,q\}$, where the fault type is a link failure. Table 2 defines the coordinate S operator.

Define the S operator between c_h and f_s as follows. Let $c_h \, \$ \, f_s = c_h$, if $a_i \, \$ \, b_i = y$ for any i; else, let $c_h \, \$ \, f_s = X \cup Y \cup Z$, where for some j, $a_j \, \$ \, b_j = t$, and

$$X = \begin{cases} \varnothing & ; \text{if } a_j \$ b_j = t \text{ for some } j \text{ and } a_i \$ b_i = t \text{ for all } i \neq j, \\ \bigcup_{i \in P} a_{n-1} \ldots a_{i+1} \alpha_i a_{i-1} \ldots a_0 & ; \text{ otherwise, where} \\ & \qquad P = \{ i \mid a_i \$ b_i = \alpha_i = 0 \text{ or } 1 \}, \end{cases}$$

$Y = a_{n-1} \ldots a_{j+1}\ 0\ a_{j-1} \ldots a_0$, and $Z = a_{n-1} \ldots a_{j+1}\ 1\ a_{j-1} \ldots a_0$. (2)

If $C = \{c_1, \ldots, c_r\}$ is a set of cubes, then let $C \, \$ \, f_s = \bigcup_{h=1}^{r} c_h \, \$ \, f_s$. Note that in the coordinate S operation, y indicates that the cube and link are disjoint, z indicates a possible overlap, and t in dimension j indicates that the faulty link is in dimension j and the subcube contains links in dimension j.

a_i \ b_i	0	1	x
0	z	y	z
1	y	z	z
x	1	0	z

Table 1. Coordinate # operation

a_i \ b_i	0	1	q
0	z	y	y
1	y	z	y
x	1	0	t

Table 2. Coordinate $ operation

Example 1. Consider a faulty link 00q (= f_1) in Q_3. For c_1 = xxx, we obtain 11t by the coordinate $ operation. From Equation (2), the list of fault-free subcubes is $c_1 \$ f_1$ = {1xx, x1x, xx1, xx0}. The first two values are all maximal subcubes in c_1 disjoint from 00x, the 1-subcube containing link 00q; the next two values correspond to c_1 split along the direction of link 00q.

Definition 3. (a) Let $f_s = b_{n-1} \ldots b_i \ldots b_0$ be the TV description of a subcube. Then $D(f_s) = \bigcup_{i \in P}$ xx ... x $\bar{a}_i$ x ... xx, where $P = \{i \mid a_i = 0 \text{ or } 1\}$.

(b) Let $f_s = b_{n-1} \ldots b_i \ldots b_0$ be the TV description of a link, where b_j = q. Then $D(f_s) = X \cup Y \cup Z$, where $X = \bigcup_{i \in P}$ xx ... x $\bar{a}_i$ x ... xx, where $P = \{i \mid i \neq j\}$, Y = xx ... x0x ... xx, and Z = xx ... x1x ... xx, where the 0 and 1 are in position j.

For example, D(101x) = {0xxx, x1xx, xx0x} and D(10q1) = {0xxx, x1xx, xx0x, xx1x, xxx0;. Set $D(f_s)$ contains all maximal subcubes disjoint from f_s.

Definition 4. pl operation. Let c_h and f_s denote subcubes, where $c_h = a_{n-1} \ldots a_i \ldots a_0$ and $f_s = b_{n-1} \ldots b_i \ldots b_0$, where $a_i \in \{0,1,x\}$ and $b_i \in \{0,1,x\}$. The coordinate *pl* operation is defined in Table 3.

Define the *pl* operation between c_h and f_s as follows.

$$c_h \ pl \ f_s = \begin{cases} \varnothing & \text{; if } a_i \ pl \ b_i = y \text{ for any } i,\ 1 \le i \le n \\ d_{n-1} \cdots d_1 d_0 & \text{; otherwise, where } d_i = a_i \ pl \ b_i, \text{ for } 1 \le i \le n \end{cases}$$

Note that *pl* returns the subcube common to c_h and f_s. If c_h and f_s are disjoint, the coordinate *pl* operation returns a y in the bit position for which c_h has a 1 and f_s has a 0 (or vice versa), and the *pl* operation returns the null set.

c_r \ f_s	0	1	x
0	0	y	0
1	y	1	1
x	0	1	x

Table 3. Coordinate *pl* operation

Examples 2a, 2b, and 2c given below, illustrate the *pl* operation and the use of it for redundancy (duplication and absorption) checking. Assume the reference subcube as X.

Example (2a)		*Example (2b)*		*Example(2c)*	
(X)	1 1 x x 0	(X)	1 1 x x 0	(X)	1 1 x x 0
(Y)	1 1 x x 0	(Z)	1 1 x 0 0	(W)	1 x x 1 0
pl: (A)	1 1 x x 0	*pl*: (B)	1 1 x 0 0	*pl*: (C)	1 1 x 1 0

Observe in Example 2a that cube A is equal to cube Y. Thus, cube Y is a subcube of cube X and is redundant. Example 2b is similar. Example 2a is a case in which Y is identical to X and Example 2b is a case in which Z is a proper subcube of X. Observe in Example 2c that cube C is not equal to either cube W or cube X. Recall that *pl* produces the subcube that is contained in both X and W. Thus, cube W is not a subcube of cube X, and vice versa, so neither cube is redundant.

4. Reconfiguration Algorithms

In this section, we present two reconfiguration algorithms based on the operators discussed in Section 3. Algorithm 2 identifies and removes the redundant terms generated in Algorithm 1. For both algorithms, C_{i+1} represents the set of fault-free subcubes after the ith fault.

Algorithm 1. [Identification of fault-free subcubes]

Input: List of subcube and/or link faults $f_1, f_2, \ldots, f_m$.
$C_1 = \{c_1\}$, where c_1 = x x x ... x. /* Q_n is initially fault-free. */
for $i = 1$ to m do
begin
if f_i represents a subcube failure
then $C_{i+1} = C_i \# f_i$
else $C_{i+1} = C_i \$ f_i$ /* f_i represents a link failure */
if $C_{i+1} = \emptyset$, then return $\emptyset$
end
Return C_{m+1}.

Theorem 1: Given a list of faulty subcubes and links in an n-dimensional hypercube, Algorithm 1 identifies all maximal dimension fault-free subcubes.

Proof: Let Ξ be the fault-free portion of a faulty n-cube. Let C_m be the set of all maximal fault-free subcubes in Ξ. Now consider a faulty subcube or link f in Ξ. Let Ξ' denote Ξ with f removed. We establish that, given C_m, Algorithm 1 produces the set C_{m+1} of all maximal fault-free subcubes in Ξ'. This will prove that Algorithm 1 applied to a succession of faults in an n-cube in turn will produce the set of all maximal fault-free subcubes of the n-cube, since Algorithm 1 begins with a fault-free n-cube.

Consider a particular subcube $c = a_{n-1} \ldots a_i \ldots a_0$ of C_m, where $a_i \in \{0,1,x\}$. We will show that given a new fault f, Algorithm 1 returns the set c' of maximal fault-free subcubes in c, that is, the set of all maximal subcubes in c that do not contain f. We describe the case in which f is a subcube; the case in which f is a link is handled similarly.

Let $f = b_{n-1} \ldots b_i \ldots b_0$ be a faulty subcube, where $b_i \in \{0,1,x\}$. Algorithm 1 computes the set $c' = c \# f$. Algorithm 1 computes the coordinate # operation between c and f. If $a_i \# b_i = y$ for any i, then c contains a 1 (0) and f contains a 0 (1) in coordinate position i, so they are disjoint and $c' = \{c\}$. If $a_i \# b_i = z$ for all i, then either $a_i = b_i$ or $b_i = x$ for each i. Consequently, c is contained completely within f, so $c' = \emptyset$.

Otherwise, c and f overlap, but neither contains the other. In these cases, $c' = \bigcup_{i \in P} a_{n-1} \ldots a_{i+1}\ \alpha_i\ a_{i-1} \ldots a_0$, where $P = \{i \mid a_i \# b_i = \alpha_i = 0 \text{ or } 1\}$. For fixed i, $a_i \# b_i = 0$ (1) if $a_i = x$ and $b_i = 1$ (0). The cube $c_i' = a_{n-1} \ldots a_{i+1}\ \alpha_i\ a_{i-1} \ldots a_0 = a_{n-1} \ldots a_{i+1}\ \overline{b_i}\ a_{i-1} \ldots a_0$ is clearly disjoint with f and contained in c. Cube c_i' is one dimension smaller than c and so is maximal. All other subcubes of c either overlap with f or are contained within an element of c'.

The argument above establishes that Algorithm 1 produces the set of all maximal fault-free subcubes within each individual element of C_m. This set, over all elements of C_m, forms C_{m+1}. It is clear that since Ξ contains all maximal fault-free subcubes before fault f, then no maximal fault-free subcubes in Ξ' are excluded from C_{m+1}. Therefore, Algorithm 1 produces all maximal fault-free subcubes within a faulty n-cube. ∎

Algorithm 1 may produce redundant terms. In what follows, we describe a general approach to produce only nonredundant terms using the D and pl operators. Shier and Whited [12] confronted a similar problem while generating cutsets from pathsets by a process called inversion. We introduce the following procedure that, given a set $E = \{E_1, E_2, \ldots, E_p\}$ of fault-free subcubes and the set $e = \{e_1, e_2, \ldots, e_r\}$ of all maximal subcubes disjoint from a new fault, produces the new set of fault-free subcubes and eliminates redundant terms in the result.

Procedure *Find_Subcubes*(E, e)
begin
  $RL1 = \emptyset$; $RL2 = \emptyset$
  For each pair (E_j, e_i), where $1 \le j \le p$ and $1 \le i \le r$, do the following
    $Z = E_j\ pl\ e_i$
    if $Z = \emptyset$, then go to the next (i, j) pair.
    if $Z \neq \emptyset$ then do
    begin
      $Y_i = 1$; $Y_j = 1$
      if $Z = e_i$, then $Y_i = 0$
      if $Z = E_j$, then $Y_j = 0$
      if $Y_i = 0$ and $Y_j = 0$
      then $e = e - e_i$; $E = E - E_j$
        $RL1 = RL1 \cup Z$, and go to the next (i, j) pair.
      else if Y_i (Y_j) = 0 and Y_j (Y_i) ≠ 0
        then $e = e - e_i$ ($E = E - E_j$)
          $RL1 = RL1 \cup Z$, and go to the next (i, j) pair.

```
            else if Y_i ≠ 0 and Y_j ≠ 0
                then RL2 = RL2 ∪ Z
        end
    For each pair (α, β), such that α ∈ RL1, β ∈ RL2, do
        begin
        perform redundancy checking on (α, β); refer to Example 2
        if β is redundant, then RL2 = RL2 − β
        end
    For each pair (α, β), such that α, β ∈ RL2, do
        begin
        perform redundancy checking on (α, β); refer to Example 2
        if α is redundant, then RL2 = RL2 − α
        if β is redundant, then RL2 = RL2 − β
        end
    RL1 = RL1 ∪ RL2
    Return RL1
end
```

Lemma 1: *Find_Subcubes*(E, e) returns a set of terms $RL1$ that contains exactly the nonredundant set of subcubes common to E and e.

Proof sketch: If cubes E_j and e_i are disjoint, then the coordinate *pl* operation between E_j and e_i returns a y in at least one position, and $Z = E_j \; pl \; e_i = \varnothing$ and so makes no contribution to the set of subcubes. Otherwise, the coordinate *pl* operation returns the term Z that describes the cube common to E_j and e_i.

The algorithm next tests whether $Z = e_i$ or $Z = E_j$. If $Z = e_i$, then this implies that each x in e_i matches with an x in E_j, but E_j may have more x's than e_i. In this case, $Y_i = 0$ and e_i is a subcube of E_j, so e_i is removed. Similarly, if $Y_j = 0$, then E_j is a subcube of e_i and is removed. Similar reasoning holds for other cases. ∎

Lemma 2: Let t_1 (t_2) be the size of $RL1$ ($RL2$). The redundancy checking in *Find_Subcubes* requires up to $t_1 t_2 + \binom{t_2}{2}$ containment checks and leaves only nonredundant residual terms in $RL2$.

A straightforward approach to redundancy checking requires as many as $\binom{t_1 \cdot t_2}{2}$ containment checks.

As an example, let links $f_1 = 0q$ and $f_2 = q1$ be faulty in Q_2. Using TV notation, $C_2 = \{1x, x1, x0\}$ and $D(f_2) = \{1x, 0x, x0\}$. Applying *Find_Subcubes*$(C_2, D(f_2))$, we get $RL1 = \{1x, x0\}$, and $RL2 = \{01, 00\}$ before redundancy checking. The redundancy checking of the procedure removes term '00' from $RL2$. Finally, we obtain three non-redundant terms $\{1x, 01, x0\}$.

The procedure below reduces the amount of computational efforts considerably over Algorithm 1. Note, the order in which the E vector is processed may appreciably affect

the total amount of work done by the algorithm. In reliability literature [13], preprocessing according to the increasing order of cardinality of terms is found to be beneficial from the complexity viewpoint. In this case, a similar sorting will help find a maximum size d-subcube and also contain the computational effort of the procedure. Thus, we obtain the following algorithm.

Algorithm 2.

Input: List of subcube and/or link faults $f_1, f_2, \ldots, f_m$.
$C_2 = D(f_1)$
for $i = 2$ to m do
begin
$e = D(f_i)$
$C_{i+1} = Find\_Subcubes(C_i, e)$
$C_{i+1} = \text{Sort}(C_{i+1})$
end
Return C_{m+1}.

Theorem 2: Given a list of faulty subcubes and links in an n-dimensional hypercube, Algorithm 2 identifies all maximal dimension fault-free subcubes.

Proof: $D(f_i)$ generates a covering for the minterms not contained in f_i. Thus $C_2 = D(f_1)$ has the effect of $C_2 = xx\ldots x \# f_1$ or $C_2 = xx\ldots x \$ f_1$, where f_1 is a subcube or link failure, respectively. By Lemma 1, $Find\_Subcubes(C_i, D(f_i))$ computes a set of terms corresponding to the maximal subcubes contained in both C_i and $D(f_i)$ with the redundant terms removed, which again has the same effect as $C_i \# f_i$ or $C_i \$ f_i$, where f_i is a subcube or link failure, respectively. Thus, Algorithm 2 computes the same result as Algorithm 1. By Theorem 1, the theorem is proved. ∎

5. Complexity Analysis

We now analyze the time complexity of the algorithms. For a given list of m faults, Algorithm 1 computes $C_{i+1} = C_i \# f_i$ or $C_i \$ f_i$, depending on whether f_i is a subcube or link fault, on each of m iterations. Assume that computing $c_r \# f_s$ for one cube c_r and for one cube f_s can be done in one time step for each resulting cube, so our object is to bound the number of cubes in C_{i+1}. We consider subcube faults only, assuming that each fault is a 0-subcube (that is, a node), as this will produce the worst case bounds. Initially, $C_1 = \{x\ x\ x\ \ldots\ x\}$. By definition of the # operator, in the worst case, C_2 may contain n cubes: $x\ x\ \ldots\ x\ \alpha_0$, $x\ x\ \ldots\ x\ \alpha_1\ x$, ..., $\alpha_{n-1}\ x\ \ldots\ x$, where $\alpha_i \in \{0, 1\}$. In the worst case, the set of cubes produced by $c_r \# f_s$ may contain at most as many cubes as there are x's in c_r, and each of those cubes will contain one less x than c_r. Hence, C_i may contain at most $n(n-1)\cdots(n-i+1) = n!/(n-i)!$ cubes, where n is the dimension of the hypercube Q_n under consideration. Actually, with n symbol positions and 3 possible symbols, $\{\ \ 1, x\}$, there are at most 3^n possible cubes. Let v be the least value of i such that $n!/(n-i)! \geq 3^n$. Note that $n(n-1)\cdots(n-i+1) < n^i$. So the time to compute m iterations in the algorithm, for $m \leq v$, is

$$\sum_{i=1}^{m} \frac{n!}{(n-i)!} < \sum_{i=1}^{m} n^i = O(n^m).$$

And the time to compute m iterations in the algorithm, for $m > v$, is

$$\sum_{i=1}^{v} \frac{n!}{(n-i)!} + \sum_{i=v+1}^{m} 3^n = O(n^v) + O((m-v)3^n) < O(m3^n).$$

In terms of N ($= |V| = 2^n$), the size of the hypercube, the time complexity for number of faults $m \leq v$ is $O(\log^m N) = o(N)$, and the time complexity for $m > v$ is $O(mN^{\beta})$, where $\beta = \log_2 3$. This improves on the time complexity $O(n(N-m)^2)$ of Sridhar and Raghavendra's algorithm [14].

If we wish instead to compute a list of only the fault-free subcubes of dimension at least $n-k$, then we can obtain a better time complexity. If $m \leq k$, then we again obtain a time complexity of $O(n^m)$. But if $m > k$, then we obtain the following time complexity.

$$\sum_{i=1}^{k} \frac{n!}{(n-i)!} + \sum_{i=k+1}^{m} \frac{n!}{(n-k)!} = O(n^k) + O((m-k)n^k).$$

This time improves on Özgüner and Aykanat's algorithm [8] that requires $O\left(mk\binom{n}{k}\right)$ time to locate the available subcubes of dimension $n-k$ or greater.

Algorithm 2 has a greater worst case time complexity, but a better expected time complexity. As noted in the discussion of Algorithm 2, the *pl* operator and equality tests are used to remove redundant terms from C_{i+1}. Thus, the number of terms in each C_i of Algorithm 2 will be fewer than the number of terms in each C_i of Algorithm 1. In the worst case, Algorithm 2 takes more time because of the time spent in redundancy checking and sorting. In particular, the ith iteration takes $O(n^{2i})$ time as opposed to $O(n^i)$ time for the other algorithms, leading to an overall time complexity of $O(n^{2m})$ for $m \leq v$ as opposed to $O(n^m)$ and still $O(m3^n)$ for $m > v$. The detection and removal of redundant terms should, however, allow improved time complexity as it will reduce the size of each C_i.

References

1. B. Becker and H. Simon, How robust is the n-cube?, *Inf. and Comput.* 77 (1988) 162-178.
2. K. V. Bhatt, An efficient approach for fault diagnosis in a Boolean n-cube array of microprocessors, *IEEE Trans. Computers* C-32 (1983) 1070-1071.
3. W. J. Dally, Performance analysis of k-ary n-cube interconnection network, *IEEE Trans. Computers* C-39 (1990) 775-785.
4. F. Harary, A survey of the theory of hypercube graphs, *Compu. Math. Applications* 15 (1988) 277-289.
5. J. Kim, C. R. Das, and W. Lin, A top-down processor allocation scheme for hypercube computers, *IEEE Trans. Parallel and Distributed Systems* 2 (1991) 20-30.
6. S. Latifi, Distributed subcube identification algorithms for reliable hypercubes, *Inf. Process. Lett.* 38 (1991) 315-321.

7. R. Miller, *Switching theory, volume I: Combinational circuits* (Wiley, New York, 1965).
8. F. Özgüner and C. Aykanat, A reconfiguration algorithm for fault tolerance in a hypercube multiprocessor, *Inf. Process. Lett.* **29** (1988) 247-254.
9. S. Rai and J. L. Trahan, ATARIC: An algebraic technique to analyse reconfiguration for fault tolerance in a hypercube, *Proc. 3rd IEEE Symp. Par. and Distr. Processing* (199?) 548-555.
10. H. K. Regbati, Fault detection in PLAs, *IEEE Design and Test* (1986) 43-50.
11. Y. Saad and M. H. Schultz, Topological properties of hypercubes, *IEEE Trans. Computers* C-37 (1988) 867-871.
12. D. R. Shier and D. E. Whited, Algorithms for generating minimal cutsets by inversion, *IEEE Trans. Reliability* R-34 (1985) 314-319.
13. S. Soh and S. Rai, CAREL: Computer aided reliability evaluator for distributed computer networks, *IEEE Trans. Parallel and Distributed Systems* **2** (1991) 199-213.
14. M. A. Sridhar and C. S. Raghavendra, On finding maximal subcubes in residual hypercubes, *Proc. 2nd Symp. Par. and Distr. Processing* (1990) 870-873.
15. M. Veeraraghavan and K. Trivedi, An improved algorithm for the symbolic reliability analysis of networks, *Proc. 9th Symposium on Reliable Distributed Systems* (1990) 34-43.

Proceedings of the ISMM
International Conference

PARALLEL AND DISTRIBUTED COMPUTING

AND SYSTEMS

Pittsburgh, PA, U.S.A.
October 1-3, 1992

Editor: R. Melhem

A Publication of
The International Society for
Mini and Microcomputers - ISMM

ISBN: 1-880843-02-1

Improved Lower Bounds on the Reliability of Hypercube Architectures

Sie Teng Soh, Suresh Rai, and Jerry L. Trahan

Department of Electrical & Computer Engineering
Louisiana State University, Baton Rouge, LA 70803

Summary & Conclusions - The hypercube topology, also known as the Boolean n-cube, has recently been used for multiprocessing systems. Several authors have analyzed the performance of hypercube-based systems. As the size and complexity of a system increases, however, the reliability aspects become equally important and should be included in the performance parameter study of the system. This paper describes algorithms for computing lower bounds on two reliability metrics, namely, terminal reliability [TR] and network reliability [NR], measures often used for packet-switching applications. The terminal (network) reliability is defined as the probability that there exists a working path connecting two (all) nodes in the stochastic graph model of the hypercube. Note, there are no known polynomial time algorithms for exact computations of either TR or NR for the hypercube, thus, lower bound computation is a better approach. The paper presents polynomial time algorithms that obtain improved lower bounds for both TR and NR than known results. Some existing techniques for TR and NR evaluation are discussed, and lower bound results are compared with previous bounds. Illustrating examples are provided to describe the proposed techniques. Our results show that for link reliability $p = 0.95$ or better, which is the case for practical hypercube-based systems, both reliability parameters are close to 1. These results further verify the robustness of the hypercube architectures under link failures.

Keywords. Combinatorics, Hypercube, Lower Bounds, Network Reliability, Terminal Reliability.

1. Introduction

The hypercube topology, also known as the Boolean n-cube or binary n-cube [4], has recently been used for multicomputer systems. Each of the 2^n nodes of an n-cube is a computer which is directly connected to n neighboring nodes. References [4,5] discuss topological properties of a hypercube graph. Performance analysis of hypercube-based systems has been addressed in [6,7]. As the size and complexity of a system increases, reliability aspects become increasingly important parameters to be included in the performance analysis of the system.

The reliability of hypercube-based multicomputer systems is generally evaluated using the following models.

1) Terminal Reliability (TR) Model: The system works as long as a specified input (node) is connected to a specified output (node).
2) Network Reliability (NR) Model [8]: The system works as long as all nodes in the system are connected.
3) Task-based Reliability Model [1,2]: The system works as long as some minimum number of connected nodes are available on the system for task execution.
4) Functional Subcube Model [3]: The system works as long as some functional minimum degree subcube exists.

This work is supported in part by the Air Force Office of Scientific Research under grant AFOSR-91-0025

The models may be evaluated by assuming that only the nodes can fail while the links are reliable, or only the links can fail while the nodes are perfect, or both nodes and links can fail. In addition, each model assumes that node / link failures are statistically independent. A conventional reliability modeling approach usually considers a stochastic graph model with failing links. In hypercube structures, this approach is warranted to verify the sturdiness of the network model of the topology. Moreover, in a hypercube, the network reliability is obvious with perfect links and failing nodes. The NR with node failures is just the probability that all nodes are operational, which is the product of the node reliabilities.

In general networks the problem of computing any of the first three measures exactly is $\mathcal{NP}$-complete [9] and so will require an unreasonable amount of computation time. It is often possible, however, to much more efficiently obtain upper and lower bounds on a reliability measure. The lower bound is of greater interest as the system will be at least this reliable.

In this paper, we investigate the TR and NR models, and consider the link failures case. We propose new techniques to improve lower bounds for both the TR and NR. These techniques produce better results and are computable in time polynomial in the order of the dimension of the hypercube.

The layout of the paper is as follows. Section 2 presents the background material which includes notations, assumptions, and discussion on earlier work done on the topic. Section 3 proposes an algorithm to obtain a tighter lower bound on TR, while Section 4 develops an algorithm for an improved bound on NR. In Section 5, we present the bounds computed for TR and NR using the new techniques, and compare the results with those obtained by previous methods.

2. Background

2.1 Notations and Assumptions

Notations

C_n	n-cube or hypercube.
N (L)	number of nodes (links) in C_n; $N = 2^n$, $L = n2^{n-1}$
p (q)	link reliability (unreliability); $p+q = 1$.
$ST(C_n)$	number of minimal spanning trees in C_n; $ST(C_n) = 2^{2^n - n - 1} \prod_{j=1}^{n} (2j)^{\binom{n}{j}}$.
Φ_k	a function representing $(1 - p^k)$
$TR(C_n, p)$	terminal reliability of C_n with link reliability p.
$NR(C_n, p)$	network reliability of C_n with link reliability p.
G_i	ith group of paths.
$P_{i,j}$	jth path of group G_i.
$RF_{i,j}$	reliability contribution of $P_{i,j}$.
R_i	reliability contribution of G_i.

Assumptions

a) A link is bidirectional.

Links are statistically identical and have the same probability of success p.

Link failures are statistically independent, and nodes are perfect.

2. Previous Work

2.1 Terminal Reliability

Let (s, t) be a pair of source and terminal nodes in an n-cube. We will compute a lower bound on TR using only shortest length paths. Without loss of generality, consider the addresses for s and t to be 0 and 2^n-1, respectively. These nodes are at diameter distance. There are $n!$ (s, t) minpaths in C_n [4]. The terminal reliability (TR) of an n-cube is the probability that there is a working path from s to t despite link failures. This probability can be computed by first enumerating the $n!$ source-terminal paths for the C_n, and then using the sum of disjoint products (SDP) technique [9] to transform it into the analogous reliability expression for the TR value. The numbers of links and paths for the cube grow exponentially with the number of nodes in C_n; hence, it is not efficient to compute TR for large C_n using the above method. Alternatively, computing a lower bound on TR offers the possibility of obtaining an important insight into the value of TR at a much better computational cost.

Of the $n!$ (s, t) minpaths, n of these are disjoint. A lower bound on TR is computed by considering only these n disjoint paths utilizing a method used to obtain the reliability of a series-parallel system [9]. Let $TR_1(C_n, p)$ be the lower bound on TR for a C_n with link reliability p by considering only its n disjoint paths. Then,

$$TR_1(C_n, p) = 1 - (1 - p^n)^n. \tag{1}$$

The model is straightforward, but rapidly deteriorates as n increases. In Section 3, we provide a tighter bound on terminal reliability for an n-cube.

2.2 Network Reliability

The network reliability (NR) or all-terminal reliability of a C_n is the probability that every node in the n-cube can communicate with every other node despite link failures. One may compute the exact value for NR of a C_n by first generating its minimal spanning trees, then in turn using these to obtain NR. An n-cube is a matroid [9], so CAREL [11] is able to compute NR from the set of minimal spanning trees, which is of size $ST(C_n)$, in time polynomial in $ST(C_n)$. Unfortunately, $ST(C_n)$ is exponential with respect to n. Thus, this technique is not advisable for large C_n. Furthermore, there is no known polynomial time algorithm for the NR problem in C_n. Obtaining a lower bound on NR is, obviously, a more practical approach. Yang *et al.* [10] have a lower bound on NR by considering the number of spanning trees in C_n. Then, they weighted each spanning tree by considering its 2^n-1 links as operational and the rest of the links set as failed. Their bound is acceptable only for very small p [10]. The reliability polynomial concept [9] may also be used for NR. It is noted in [8], however, the resultant lower bound on NR is not tight. Furthermore, the method based on this concept is polynomial time only in terms of the number of links, and so is exponential in the dimension of the n-cube. Bulka and Dugan [8] got a lower bound on the NR of an n-cube from a lower bound on the reliability of an $(n-1)$-cube, which in turn is obtained from one on an $(n-2)$-cube, and so on, until the cube is small enough (base cube) that its reliability can be evaluated exactly. Note, Bulka and Dugan used a C_2 as the base case, for which the exact reliability is given as:

$$NR(C_2, p) = 4p^3(1-p)+p^4.$$

Bulka and Dugan's algorithm [8] computes the lower bound on NR of a C_n in three steps. In the following, all 2^{n-1} links connecting two C_{n-1}'s are termed as exterior links.

Step 1. Both $(n-1)$-cubes are operating and i exterior links operate, $1 \le i \le 2^{n-1}-2$.

$$T1 = NR(C_{n-1}, p)^2 \cdot \sum_{i=1}^{2^{n-1}-2} \binom{2^{n-1}}{i} p^i q^{2^{n-1}-i}$$

Step 2. At least one $(n-1)$-cube is operating, and one exterior link is failed.

$$T2 = [2 \cdot NR(C_{n-1}, p) \cdot (1-q^{2^{n-1}}) - NR(C_{n-1}, p)^2] \cdot 2^{2^{n-1}} p^{2^{n-1}} q$$

Step 3. All 2^{n-1} exterior links operate.

$$T3 = NR(C_{n-1}, p^2+2pq)\, p^{2^{n-1}}$$

The lower bound on network reliability of an n-cube is obtained as:

$$NR(C_n, p) = T1 + T2 + T3. \tag{2}$$

Computing $T1$ in Step 1 takes time exponential in the dimension of the cube, and thus is useful only for dimensions $n \le 9$. For large n, reference [8] views an n-cube as a 1-cube with two (2^{n-1})-supernodes connected by one (2^{n-1})-link. Reference [8] next generalizes the approach by considering the n-cube as a 2-cube whose four nodes are each $(n-2)$-cubes, or a 3-cube whose eight nodes are each $(n-3)$-cubes, and so on. In general, an n-cube can be viewed as an $(n-k)$-cube whose 2^{n-k} nodes are each k-cubes which are connected by 2^{n-k} (2^k)-links. Therefore, if the original link reliability is p, then the new link reliability is $1-(1-p)^{2^k}$, which is the probability that at least one of the 2^k links is operating. Using this approach, Bulka and Dugan obtained another lower bound on network reliability of an n-cube as

$$NR(C_n, p) = NR(C_k, p)^{2^{n-k}} \cdot NR(C_{n-k}, 1-(1-p)^{2^k}). \tag{3}$$

Note, for large k and p ($k \ge 2$ and $p \ge 0.9$), the second term of the right hand side in Equation (3) is approximately equal to 1. Thus, a simplified version of (3) is given as:

$$NR(C_n, p) = NR(C_k, p)^{2^{n-k}} \tag{4}$$

Comparing Equation (3) or (4) with Equation (2), it is clear that Equation (2) provides a tighter bound.

In what follows, we suggest an improvement on Equation (2). First, using the simple combinatorial identity $\sum_{i=0}^{2^{n-1}} \binom{2^{n-1}}{i} p^i q^{2^{n-1}-i} = 1$, we express $T1$ as:

$$T1 = NR(C_{n-1}, p)^2 \cdot (1-q^{2^{n-1}}-p^{2^{n-1}}-2^{n-1}p^{2^{n-1}}q). \tag{5}$$

Note, Equation (5) is polynomial in the dimension of the cube, and hence, we need not use an inferior lower bound obtained by Equation (3) or (4). Second, to improve reliability contribution of the events in Step 3, we alternatively consider the following situations:

Step 3a. At least one $(n-1)$-cube operates.

$$T3a = [NR(C_{n-1}, p)^2 + 2NR(C_{n-1}, p)(1-NR(C_{n-1}, p))]\, p^{2^{n-1}}.$$

Step 3b. Both $(n-1)$-cubes fail, but if we contract each pair of congruent nodes reducing the n-cube into an $(n-1)$-cube, the combined links form a connected $(n-1)$-cube. The probability is given as

$$T3b = [2 \cdot (1 - NR(C_{n-1}, p))^2 \cdot NR(C_{n-1}, p)]\, p^{2^{n-1}}.$$

However, even with the suggested improvement, the resultant lower bound is still not tight for $p < 0.9$ and $n > 10$. In Section 4 we present a better lower bound on network reliability which gives a significant improvement over the results based on the discussed method.

3. Terminal Reliability - An Improved Bound

To improve the lower bound $TR_1(C_n, p)$, we consider $n(n-2)$ minpaths, selected and ordered using a routing method described below, and then use a Boolean technique to compute the reliability value. Note, the selected $n(n-2)$ minpaths include the n disjoint paths used for generating $TR_1(C_n, p)$, so our lower bound is tighter. Furthermore, the proposed method uses the minpaths to analytically calculate the reliability without enumerating them. Let $TR_2(C_n, p)$ be the terminal reliability obtained from these $n(n-2)$ paths. Consider the minpaths to be in $(n-2)$ groups, each of which consists of n paths. For $i = 1, 2, \cdots, n-2$, and $j = 1, 2, \cdots, n$, let $P_{i,j}$ represent the jth path in a group of paths G_i. In what follows we present the routing algorithm that enumerates the $n(n-2)$ minpaths in C_n.

3.1 Routing Algorithm

Any minpath from source node 0 to terminal node 2^n-1 traverses n links in n different dimensions. Number the dimensions by 0 to $n-1$. Let $P_{i,j}$ be $d_0 d_1 \cdots d_{n-1}$, where $\{d_0, d_1, \cdots, d_{n-1}\} = \{0, 1, \cdots, n-1\}$, denote the path from node 0 to node 2^n-1 obtained by first traversing a link in dimension d_0, then traversing a link in dimension d_1, $\cdots$, then traversing a link in dimension d_{n-1}.

Algorithm

1) $P_{1,1} = 0\ 1\ 2\ 3 \ldots (n-1)$
   for $j = 2$ to n do begin
      Path $P_{1,j}$ is obtained by traversing dimensions in the order given by a left rotate by one of $P_{1,j-1}$.
   end;
2) for $i = 2$ to $n-2$ do begin
      for $j = 1$ to n do begin
         $P_{i,j}$ has the same first traversed dimension as $P_{1,j}$.
         The last $i-1$ traversed dimensions of $P_{i,j}$ are the same as those in $P_{i-1,j}$. The remaining $n-i$ dimensions (that is the $2nd$ to $(n-i+1)th$ dimensions traversed) are given by a left rotate by one of the dimensions in the same positions in $P_{i-1,j}$.
      end;
   end;

The algorithm generates $n(n-2)$ paths with the following properties.

Property 1. Paths $P_{i,j}$ and $P_{i,k}$ are link disjoint, for $j \neq k$.
Property 2. Paths $P_{i,j}$ and $P_{k,j}$ have $(i+1)$ common links, for $i < k$.
Property 3. Paths $P_{i,j}$ and $P_{k,l}$ are link disjoint, for $j \neq l$.

Observe that G_1 comprises the n disjoint paths considered for $TR_1(C_n, p)$. The $TR_2(C_n, p)$ is computed analytically by using the concept of Boolean techniques [9,11].

Example 1. To illustrate Properties 1 through 3, consider a C_6 with source and terminal nodes as $s = 000000$ and $t = 111111$, respectively. Using above algorithm, the various 6(6-2) minpaths grouped and ordered are given as follows, where the path is listed to the left of the arrow and the nodes visited (excluding the source and terminal nodes) are listed to the right of the arrow.

Group G1:

$P_{1,1}$: 0 1 2 3 4 5 -> 000001-000011-000111-001111-011111
$P_{1,2}$: 1 2 3 4 5 0 -> 000010-000110-001110-011110-111110
$P_{1,3}$: 2 3 4 5 0 1 -> 000100-001100-011100-111100-111101
$P_{1,4}$: 3 4 5 0 1 2 -> 001000-011000-111000-111001-111011
$P_{1,5}$: 4 5 0 1 2 3 -> 010000-110000-110001-110011-110111
$P_{1,6}$: 5 0 1 2 3 4 -> 100000-100001-100011-100111-101111

Group G2:

$P_{2,1}$: 0 2 3 4 1 5 -> 000001-000101-001101-011101-011111
$P_{2,2}$: 1 3 4 5 2 0 -> 000010-001010-011010-111010-111110
$P_{2,3}$: 2 4 5 0 3 1 -> 000100-010100-110100-110101-111101
$P_{2,4}$: 3 5 0 1 4 2 -> 001000-101000-101001-101011-111011
$P_{2,5}$: 4 0 1 2 5 3 -> 010000-010001-010011-010111-110111
$P_{2,6}$: 5 1 2 3 0 4 -> 100000-100010-100110-101110-101111

Group G3:

$P_{3,1}$: 0 3 4 2 1 5 -> 000001-001001-011001-011101-011111
$P_{3,2}$: 1 4 5 3 2 0 -> 000010-010010-110010-111010-111110
$P_{3,3}$: 2 5 0 4 3 1 -> 000100-100100-100101-110101-111101
$P_{3,4}$: 3 0 1 5 4 2 -> 001000-001001-001011-101011-111011
$P_{3,5}$: 4 1 2 0 5 3 -> 010000-010010-010110-010111-110111
$P_{3,6}$: 5 2 3 1 0 4 -> 100000-100100-101100-101110-101111

Group G4:

$P_{4,1}$: 0 4 3 2 1 5 -> 000001-010001-011001-011101-011111
$P_{4,2}$: 1 5 4 3 2 0 -> 000010-100010-110010-111010-111110
$P_{4,3}$: 2 0 5 4 3 1 -> 000100-000101-100101-110101-111101
$P_{4,4}$: 3 1 0 5 4 2 -> 001000-001010-001011-101011-111011
$P_{4,5}$: 4 2 1 0 5 3 -> 010000-010100-010110-010111-110111
$P_{4,6}$: 5 3 2 1 0 4 -> 100000-101000-101100-101110-101111

3.2 Boolean Techniques Concept

Boolean techniques for reliability evaluation start with a sum of products expression for pathsets and convert it into an equivalent sum of disjoint products (SDP) expression [11]. In the SDP form, an UP or logical success (DOWN or failure) state of a link x is replaced by link reliability p (unreliability q), and the Boolean sum (product) by the arithmetic sum (product). In other words, the SDP expression is interpreted directly as an equivalent probability expression of terminal reliability. If F_i represents a path identifier (an UP state of a link in a path P_i has 1 in F_i, while a *don't care* is represented by 0), the sum of products expression F is given by:

$$F = \bigcup_{i=1}^{n} F_i \tag{6}$$

where n denotes the number of minpaths between (s, t) node pair in G(V,E). Equation (6) is modified either canonically or conservatively to generate the equivalent SDP expression, F(disjoint). The conservative modification is usually preferred, since it is more efficient compared with canonical modification, where 2^l events are required to determine F(disjoint). (l is the number of links in the network.) A simple way to generate mutually disjoint events in Equation (6) is as follows:

$$F_1 + F_2\overline{F_1} + F_3\overline{F_1}\,\overline{F_2} + \cdots + F_n\overline{F_1}\,\overline{F_2}\cdots\overline{F_{n-1}}. \tag{7}$$

where $\overline{F_i}$ denotes a DOWN event of path P_i. The probability of UP (operational) for an ith term $F_i\,\overline{F_1}\,\overline{F_2}\cdots\overline{F_{i-1}}$ can be evaluated using conditional probability and standard Boolean operations as:

$$\Pr(F_i)\cdot\Pr(\overline{F_1}\cdot\overline{F_2}\cdots\overline{F_{i-1}} \mid F_i) = \Pr(F_i)\prod_{j=1}^{i-1}\Pr(E_j).$$

Here, E_j represents a conditional cube [11] and defines conditions for a path identifier F_j DOWN given F_i UP (operational). For the equality to hold good, E_j's must have non-redundant and mutually disjoint terms. The probability of the first event $\Pr(F_i)$ can be determined in a straightforward manner since failures are assumed to be statistically independent. The various terms within E_j's will, in general, not be disjoint [9,11]. This necessitates making E_j's mutually disjoint before we generate the equivalent probability expression.

3.3 Improving Lower Bound on Terminal Reliability

(a) Boolean Approach

Let $F_{i,j}$ be the path identifier for path $P_{i,j}$. The sum of products expression F for the $n(n-2)$ minpaths is given as:

$$F = \bigcup_{i,j} F_{i,j} \;;\; \text{where } 1 \le i \le n-2,\ 1 \le j \le n. \tag{8}$$

Using Equation (7), the SDP form for (8) is obtained as:

$$\{F_{1,1} + F_{1,2}\overline{F_{1,1}} + \cdots + F_{1,n}\overline{F_{1,1}}\cdots\overline{F_{1,n-1}}\} + \cdots + \{F_{n-2,n}\overline{F_{1,1}}\cdots\overline{F_{1,n}}\cdots\overline{F_{n-2,n-1}}\}. \tag{9}$$

Note, the ith n terms in Equation (9) give the reliability contribution of the paths in G_i. Let R_i denote the reliability contribution of G_i. The lower bound on reliability of the C_n, $TR_2(C_n, p)$, is given as:

$$TR_2(C_n, p) = \sum_{i=1}^{n-2} R_i. \tag{10}$$

By Equation (1) and Property 1, $R_1 = 1 - (1-p^n)^n$. For $i \ge 2$ and $1 \le j \le n$, the disjoint expression of a term $F_{i,j}\,\overline{F_{1,1}}\cdots\overline{F_{i-1,n}}\cdots\overline{F_{i,j-1}}$ is evaluated using conditional probability, standard Boolean operation, and Properties 1 through 3 as:

$$\Pr(F_{i,j})\cdot \tag{11}$$

$$[\Pr(\overline{F_{i-1,j}}\,\overline{F_{i-2,j}}\cdots\overline{F_{1,j}} \mid F_{i,j})]\cdot \tag{12}$$

$$[\Pr(\overline{F_{i,j-1}}\,\overline{F_{i-1,j-1}}\cdots\overline{F_{1,j-1}})\cdots\Pr(\overline{F_{1,3}}\,\overline{F_{i-1,2}}\cdots\overline{F_{1,2}})] \tag{13}$$

$$[\Pr(\overline{F_{i-1,j+1}}\,\overline{F_{i-2,j+1}}\cdots\overline{F_{1,j+1}})\cdots\Pr(\overline{F_{i-1,n}}\,\overline{F_{1,n}}\cdots\overline{F_{1,1}})] \tag{14}$$

Equation (11) is easily computed as $\Pr(F_{i,j}) = p^n$. Let $P_a - P_b$ denote the portion of P_a remaining after any links in common with P_b are removed. For any $k, l < j$ and $k \neq l$, $P_{k,j}$-$P_{l,j}$ and $P_{l,j}$-$P_{l,j}$ are link disjoint, so Equation (12) is computed as:

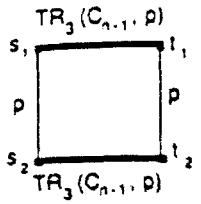


Figure 1. A 2-model of n-cube for TR problem

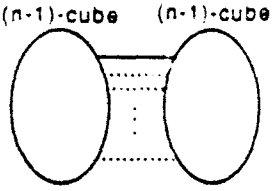


Figure 2. Two sections of n-cube

$$\Pr(\overline{F_{i-1,j}}F_{i,j}) \cdot \Pr(\overline{F_{i-2,j}}F_{i,j}) \cdots \Pr(\overline{F_{1,j}}F_{i,j}). \quad (15)$$

By Property 2, the reliability contribution of (15) is given as:

$$\prod_{r=1}^{i-1} (1 - p^{a-1-r}). \quad (16)$$

Utilizing the properties of the selected paths, each probability component of (13) has equal value. Let MG_i denote the probability of each of them, i.e.,

$$MG_i = \Pr(\overline{F_{i,k}}\,\overline{F_{i-1,k}} \cdots \overline{F_{1,k}}), \quad 1 \le k < j.$$

By Property 2, and recursively utilizing a Boolean identity $\overline{ab}A_2 \cdot \overline{ab}B_2 = \overline{ab} + abA_2 \cdot \overline{B}_2$, we obtain:

$$MG_i = \Phi_2 + p^2\Phi_{a-2}(q + p\Phi_{a-3}(\cdots (q + p\Phi_{a-i-1})(q + p\Phi_{a-i}^2))\cdots)). \quad (17)$$

Similarly, (14) is computed as MG_{i-1}. Thus, the reliability contribution of path $P_{i,j}$ is given as:

$$RP_{i,j} = p^a \prod_{r=1}^{i-1}(1 - p^{a-1-r})\,[(MG_i)^{j-1}(MG_{i-1})^{a-j}]. \quad (18)$$

where $MG_1 = \Phi_a$ and $MG_2 = \Phi_2 + p^2\Phi_{a-2}^2$. Thus, the reliability contribution of G_i is obtained as:

$$R_i = p^a \prod_{r=1}^{i-1}(1 - p^{a-1-r}) \sum_{j=1}^{a}[(MG_i)^{j-1}(MG_{i-1})^{a-j}]. \quad (19)$$

Example 2. Consider a C_6, $R_1 = 1 - (1 - p^6)^6$, $MG_1 = \Phi_6 = 1 - p^6$, and $MG_2 = (1 - p^2) + p^2(1 - p^{6-2})^2$. By Equation (17) we obtain: $MG_3 = \Phi_2 + p^2\Phi_{6-2}(q + p\Phi_{6-3}^2)$, $MG_4 = \Phi_2 + p^2\Phi_{6-2}(q + p\Phi_{6-3}(q + p\Phi_{6-4}^2))$. Utilizing Equation (19), the reliability contributions R_i's are obtained as:

$$R_2 = p^6 \prod_{r=1}^{1}(1 - p^{6-1-r}) \sum_{j=1}^{6}(MG_2)^{j-1}(MG_1)^{6-j}.$$

$$R_3 = p^6 \prod_{r=1}^{2}(1 - p^{6-1-r}) \sum_{j=1}^{6}(MG_3)^{j-1}(MG_2)^{6-j}.$$

$$R_4 = p^6 \prod_{r=1}^{3}(1 - p^{6-1-r}) \sum_{j=1}^{6}(MG_4)^{j-1}(MG_3)^{6-j}.$$

For $p = 0.9$, one gets $MG_1 = 0.468559$, $MG_2 = 0.285797$, $MG_3 = 0.236268$, and $MG_4 = 0.224167$. We also obtain $R_1 = 0.989418$, $R_2 = 0.010038$, $R_3 = 0.000371$, and $R_4 = 0.000037$, and hence Equation (10) gives $TR_2(C_6, 0.9) = 0.999863$.

(b) A 2-cube Model for TR

Consider the C_a constructed of two C_{a-1}'s that are connected by 2^{a-1} links. We call these links as exterior links, and the links within each C_{a-1} as interior links.

Theorem 1. Given a bound on TR for C_{a-1}, $TR_2(C_{a-1}, p)$, a 2-cube based lower bound on TR is expressed as:

$$TR_2(C_a, p) = 2p \cdot TR_2(C_{a-1}, p) - (pTR_2(C_{a-1}, p))^2.$$

Proof. The lower bound on TR of the C_a is computed by including only two exterior links, i.e., a link that connects the nodes 0 of the two $(n-1)$-cubes, and a link which connects the nodes $2^{a-1}-1$ in the subcubes (see Figure 1). Note, the (s, t) for the n-cube in the figure is (s_1, t_2), and the two exterior links considered are (s_1, s_2), (t_1, t_2) with link reliability p. The probability of (s_1, t_1) or (s_2, t_2) connectivity is given as the lower bound on TR for C_{a-1}. Thus, (s_1, t_2) connectivity is given as the TR for a C_2 with link reliability for (s_1, t_1), (s_2, t_2) given as $TR_2(C_{a-1}, p)$ and the other two links have probability of success p □

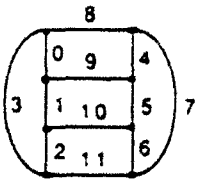


Figure 3. A 3-cube C_3

Theorem 2. The lower bound on terminal reliability for C_a, $TR(C_a, p)$, is computed as

$$TR(C_a, p) = 2p \cdot TR'(C_{a-1}, p) - (pTR'(C_{a-1}, p))^2,$$

where $TR'(C_{a-1}, p) = max\{TR_2(C_{a-1}, p), TR_3(C_{a-1}, p)\}$.

Proof. Obvious. □

The equations for lower bound on TR in Equation (10), Theorems 1 and 2 can each be computed in time polynomial in the dimension of the cube.

4. Improved Lower Bound on Network Reliability

Let us consider an n-cube C_a as two $(n-1)$-cubes connected by 2^{a-1} links. We evaluate the lower bound on reliability of a C_a from the known lower bound on reliability of a C_{a-1}. The proposed algorithm divides the problem into three mutually disjoint cases. In addition, the events in each case are also mutually disjoint among themselves. Thus, the lower bound on reliability can be calculated as the sum of the lower bounds on reliability obtained from the following three cases.

CASE 1: Both $(n-1)$-cubes and only one exterior link operate.

The graph model for Case 1 is shown in Figure 2. Since both C_{a-1}'s operate, only one good exterior link is needed to make the two subcubes combine into a connected C_a. The disjoint expression of Case 1 is given as:

$$NR(C_{a-1}, p)^2 \cdot (p + pq + pq^2 + pq^3 + \cdots + pq^{2^{a-1}-1}). \quad (20)$$

The events in Case 1 are all mutually disjoint since we consider the operating exterior links one at a time, i.e., an exterior link operates when the links previously considered good fail. Since $0 \le p, q \le 1$, Equation (20) can be reduced to:

$$NR(C_{a-1}, p)^2 \cdot (1 - q^{2^{a-1}}). \quad (21)$$

Note, Case 1 includes $ST_{CASE1}(C_a) = 2^{a-1} \cdot ST(C_{a-1}) \cdot ST(C_{a-1})$ minimum spanning trees.

CASE 2: All 2^{a-1} exterior links operate.

In Case 2, a C_a is connected if at least one C_{a-1} operates. When all exterior links are good, the congruent nodes in the two C_{a-1}'s can be combined to form an $(n-1)$-cube, C'_{a-1}, in which each pair of adjacent nodes is connected by double links. Consider the C_3 shown in Figure 3. Figure 4a gives the graph representation for Case 2. The reliability of the C'_{a-1} is given as the lower bound on reliability of a C_{a-1} with link reliability $p^2 + 2pq$ $(= 1 - q^2)$, since the C'_{a-1} is connected when it contains at least one working spanning tree, which may have links from either or both C_{a-1}'s. Thus, for $p' = 1 - q^2$, we have the following expression.

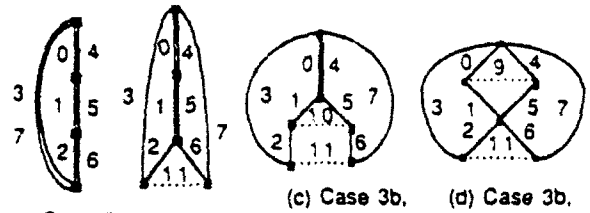
$$p^{2^{a-1}} \cdot (NR(C_{a-1}, p') - NR(C_{a-1}, p)^2) \quad (22)$$

Events in Case 2 are mutually disjoint. In addition, Case 2 is mutually disjoint from Case 1. The events considered in $NR(C_{a-1}, p')$ include the possibilities that both subcubes operate, so we subtract $NR^2(C_{a-1}, p)$ to make Case 2 disjoint from Case 1. The number of minimum spanning trees used in Case 2, $ST_{CASE2}(C_a)$, is $ST(C_{a-1})\,2^{(2^{a-1}-1)}$.

CASE 3: $2 \le i \le 2^{a-1}-1$ exterior links and one $(n-1)$-cube operate

Case 3 is intractable for large C_a's, so we compute a lower bound on reliability.

Figure 4. Merging nodes in Cube C_3 for:



(a) Case 2 (b) Case 3a (c) Case 3b, distance 1 (d) Case 3b, distance 2

CASE 3A for $i = 2^{n-1}-1$

Figure 4b depicts the graph representation of Case 3A for a C_3. As an example, the figure shows an event in which links 8, 9, and 10 operate while link 11 fails. Let an isolated node be defined as a node in a disconnected subcube C_{n-1} which has a failed exterior link as one of its links. The C_3 is connected if one C_2 cube operates and the isolated node is connected by at least one of its interior links (links 6 and 7). However, we also have included the case in which the two $(n-1)$-cubes operate. To make this case disjoint with Case 1, we subtract $NR(C_{n-1}, p)^2$. The reliability contribution for this case is given as:

$$2 \cdot 2^{n-1} \cdot p^{2^{n-1}-1} \cdot q \cdot (NR(C_{n-1}, p) \cdot (1-q^{n-1}) - NR(C_{n-1}, p)^2). \quad (23)$$

Case 3A considers $ST_{CASE3A}(C_n) = ST(C_{n-1}) \cdot (n-1) \cdot 2^n$ minimal spanning trees.

CASE 3B for $i = 2^{n-1}-2$

The two isolated nodes can be at a distance one or more. We consider the reliability expression for this case in two groups.

The two nodes are at distance 1: Figure 4c presents the graph model of this case for C_3. The figure shows the events when links 8 and 9 operate, while links 10 and 11 fail. The lower bound expression for this case is:

$$2 \cdot L' \cdot p^{2^{n-1}-2} \cdot q^2 \cdot (NR(C_{n-1}, p) \cdot (1-q^{2(n-1)-2}) \cdot p + q \cdot (1-q^{n-2})^2) - NR(C_{n-1}, p)^2)), \quad (24)$$

where L' is the number of links in C_{n-1}.

Since the two nodes are at distance 1, there is a link connecting them. First, consider the link operational. Thus, the two nodes can be merged into a node X. The C_n is connected if node X is connected by at least one of its interior links. Second, we consider the connecting link as failed. In this case, each of the two nodes should individually be connected by at least one of its interior links. However, we have also considered the case in which both C_{n-1}'s operate. Thus, to make this case disjoint from Case 1, we subtract $NR(C_{n-1}, p)^2$. This case considers $ST(C_{n-1})\,(n^3-3n^2+2n)\,2^{n-1}$ minimal spanning trees.

The two nodes are of a distance of more than 1: Figure 4d presents the graph model of this case for C_3. The figure shows the events in which links 8 and 10 operate, while links 9 and 11 fail. The lower bound expression for this case is:

$$2 \cdot \left(\left\lceil \frac{2^{n-1}}{2^{n-1}-2} \right\rceil - L' \right) \cdot p^{2^{n-1}-2} \cdot q^2 \cdot (NR(C_{n-1}, p) \cdot (1-q^{n-1})^2 - NR(C_{n-1}, p)^2). \quad (25)$$

When the two isolated nodes are at distance of more than one from each other, the n-cube is connected when each of them is connected by at least one of its interior links. Again, we have considered the case in which both $(n-1)$-cubes operate which accounts for the subtraction. The number of minimal spanning trees considered in this case is $ST(C_{n-1})\,(n-1)^2\,(2^{n-1}-n)\,2^{n-1}$ and thus, overall, Case 3B uses $ST_{CASE3B}(C_n) = ST(C_{n-1})\,2^{n-1}(2^{n-1}(n^2-2n+1)-n^2+n)$ minimal spanning trees.

CASE 3C for $2 \le i \le 2^{n-1}-3$

The problem of obtaining an exact expression for this case becomes intractable, especially for large n. However, one can get a lower bound on the reliability contribution of this case by considering a lower bound on the minimal spanning trees that have not been included in the reliability evaluated so far, and take a lower bound on reliability for each of them, i.e., multiply the number by $p^{N-1} q^d$, where $d = L-N+1$. Thus, a lower bound on reliability expression for Case 3C is:

$$(ST(C_n) - ST_{CASE1}(C_n) - ST_{CASE2}(C_n) - ST_{CASE3A}(C_n) - ST_{CASE3B}(C_n)) \cdot p^{N-1} \cdot q^d \quad (26)$$

Combining all cases,

$$NR(C_n, p) \ge (21) + (22) + (23) + (24) + (25) + (26). \quad (27)$$

The equation for lower bound on NR in Equation (27) can be computed in time polynomial in the dimension of the cube.

Table 1 shows the comparisons of the exact reliability of C_3 for the cases described obtained by CAREL [11], and the lower bound results produced by the proposed algorithm.

5. Results and Comparisons

Tables 2 to 5 show the bounds for TR_1, TR_2, TR_3, and the tightest lower bound TR for various C_n's, $n = 3, 4, \ldots, 16$ and various values of p. For $n = 3$, we have exact reliability values. As expected, TR_2 is always tighter than TR_1 since the minpaths considered in TR_1 are only a subset of the ones for TR_2. The lower bound obtained by the 2-cube model, TR_3, performs worse than TR_2 for $p > 0.8$. However, TR_3 is tighter than TR_2 for $p = 0.6$ as shown in Table 3. For $0.6 < p \le 0.8$, TR_2 is tighter than TR_3 for some C_n's. As shown in Tables 2 to 5, TR (computed by Theorem 2) always provides the best bounds for any values of p and n. As the results show, for $n \le 16$ and $p \le 0.6$, TR_3 is always as tight as TR, and thus it is suggested to use the 2-cube model for TR. On the contrary, for $p > 0.8$, TR_2 is equal to TR for $n = 3, 4, \ldots, 16$, and hence, Equation (10) is sufficient to obtain TR.

Table 6 presents the comparisons of the best known lower bounds on NR obtained in [8] with the new lower bounds generated by our proposed technique for $n = 3, 4, \ldots, 16$. Note, we have used the modified Bulka and Dugan algorithm [8], i.e., by replacing $T1$ with Equation (5). As shown in the table, our lower bounds are tighter than the Bulka and Dugan bounds, especially for $n > 10$ and $p < 0.9$.

REFERENCES

[1] W. Najjar, J.L. Gaudiot, "Reliability and Performance Modeling of Hypercube-Based Multiprocessors", *Computer Performance and Reliability* (edited by G. Iazeolla, *et al.*), 1988, pp. 305-320.

[2] J. Kim, C.R. Das, W. Lin, and T. Feng, "Reliability Evaluation of Hypercube Multicomputers", *IEEE Trans. Reliability*, vol. 38, April 1989, pp. 121-129.

[3] S. Abraham, K. Padmanabhan, "Reliability of the Hypercube", *1988 International Conference on Parallel Processing*, pp. 90-94.

[4] Y. Saad, M.H. Schultz, "Topological Properties of Hypercubes", *IEEE Trans. Computers*, vol. 37, July 1988, pp. 867-872.

[5] F. Harary, J.P. Hayes, and H. Wu, "A Survey of the Theory of Hypercube Graphs", *Comput. Math. Applic.*, vol. 15, 1988, pp. 277-289.

[6] J.T. Blake, K. Trivedi, "Multistage Interconnection Network Reliability", *IEEE Trans. Computer*, vol. 38, Nov. 1989, pp. 1600-1604.

[7] S. Rai, D.P. Agrawal, *Advances in Distributed System Reliability*, IEEE Computer Society Press, 1990. (Tutorial Text)

[8] D. Bulka, J.B. Dugan, "A Lower Bound on the Reliability of an n-Dimensional Hypercube", *Proc. 9th Symposium on Reliable Distributed Systems*, 1990, pp. 44-53.

[9] C.J. Colbourn, *The Combinatorics of Network Reliability*, Oxford University Press, New York, 1987.

[10] C.S. Yang, J.F. Wang, J.Y. Lee, and F.T. Boesch, "Graph Theoretic Reliability Analysis for the Boolean n-cube Networks", *IEEE Trans. Circuits and Systems*, vol. 35, Sept. 1988, pp. 1175-1179.

[11] S. Soh, S. Rai, "CAREL: Computer Aided RELiability Evaluator for Distributed Computer Systems", *IEEE Trans. Parallel and Distributed Systems*, vol. 2 April 1991, pp. 199-213.

[12] R.M. Ahmed, J.L. Trahan, "Two-Terminal Reliability of Hypercubes", *Proc. Southeastcon'91*, 1991, pp. 427-431.

Table 1.
Reliability for C_3 - Exact versus Lower Bound for $p = 0.9$

CASE	#Spanning Trees		Reliability	
	Exact	Lower Bound	Exact	Lower Bound
1	64	64	0.898045	0.898045
2	32	32	0.066445	0.066445
3a	64	64	0.023380	0.023379
3b	160	160	0.002488	0.002487
3c	64	64	0.000306	0.000306

Table 2.
Lower Bounds Terminal Reliability for n-cubes for p = 0.85

n-cube	TR_1	TR_2	TR_3	TR
3	0.985002	0.985002	0.985002	0.985002
4	0.947798	0.987683	0.973513	0.987683
5	0.946725	0.995617	0.970239	0.995617
6	0.941615	0.998009	0.969271	0.998009
7	0.933169	0.998899	0.968982	0.998899
8	0.921529	0.999254	0.968895	0.999254
9	0.906632	0.999371	0.968869	0.999371
10	0.888356	0.999352	0.968861	0.999352
11	0.866609	0.999221	0.968859	0.999221
12	0.841372	0.998965	0.968858	0.998965
13	0.812733	0.998552	0.968858	0.998552
14	0.780894	0.997929	0.968858	0.997929
15	0.746176	0.997030	0.968858	0.997030
16	0.709001	0.995778	0.968858	0.995778

Table 3.
Lower Bounds Terminal Reliability for n-cubes for p = 0.6

n-cube	TR_1	TR_2	TR_3	TR
3	0.703912	0.703912	0.703912	0.703912
4	0.426048	0.615492	0.666317	0.666317
5	0.332856	0.624125	0.639749	0.639749
6	0.249246	0.591016	0.620358	0.620358
7	0.180245	0.533064	0.605886	0.605886
8	0.126730	0.462511	0.594908	0.594908
9	0.087128	0.387797	0.586480	0.586480
10	0.058847	0.315047	0.579951	0.579951
11	0.039192	0.248501	0.574858	0.574858
12	0.025811	0.190725	0.570863	0.570863
13	0.016846	0.142791	0.567717	0.567717
14	0.010915	0.104571	0.565232	0.565232
15	0.007030	0.075123	0.563263	0.563263
16	0.004504	0.053085	0.561700	0.561700

Table 4.
Lower Bounds Terminal Reliability for n-cubes for p = 0.7

n-cube	TR_1	TR_2	TR_3	TR
3	0.865797	0.865797	0.865797	0.865797
4	0.666554	0.834887	0.844810	0.844810
5	0.601495	0.869762	0.833019	0.869762
6	0.528102	0.875903	0.826205	0.875903
7	0.452070	0.865198	0.822206	0.865198
8	0.378122	0.842149	0.819837	0.844479
9	0.309758	0.808747	0.818427	0.832830
10	0.249144	0.766476	0.817585	0.826095
11	0.197228	0.716753	0.817081	0.822141
12	0.154017	0.661019	0.816779	0.819799
13	0.118887	0.600801	0.816598	0.818404
14	0.090877	0.537781	0.816489	0.817571
15	0.068895	0.473799	0.816424	0.817072
16	0.051868	0.410767	0.816385	0.816774

Table 5.
Lower Bounds Terminal Reliability of n-cubes for p = 0.95

n-cube	TR_1	TR_2	TR_3	TR
3	0.999617	0.999617	0.999617	0.999617
4	0.998816	0.999873	0.997463	0.999873
5	0.999408	0.999987	0.997253	0.999987
6	0.999654	0.999998	0.997232	0.999998
7	0.999773	1.000000	0.997230	1.000000
8	0.999835	1.000000	0.997230	1.000000
9	0.999871	1.000000	0.997230	1.000000
10	0.999892	1.000000	0.997230	1.000000
11	0.999904	1.000000	0.997230	1.000000
12	0.999911	1.000000	0.997230	1.000000
13	0.999914	1.000000	0.997230	1.000000
14	0.999914	1.000000	0.997230	1.000000
15	0.999912	1.000000	0.997230	1.000000
16	0.999907	1.000000	0.997230	1.000000

Table 6.

Lower Bounds Network Reliability for n-cubes

n-cube	Link Reliability = 0.9		Link Reliability = 0.95	
	Bulka-Dugan	New Bound	Bulka-Dugan	New Bound
3	0.987870	0.990663	0.998694	0.998909
4	0.994361	0.997593	0.999754	0.999865
5	0.994326	0.998689	0.999885	0.999963
6	0.990421	0.998311	0.999882	0.999977
7	0.981114	0.996755	0.999801	0.999969
8	0.962586	0.993522	0.999606	0.999941
9	0.926571	0.987085	0.999211	0.999883
10	0.858534	0.974337	0.998423	0.999766
11	0.737080	0.949333	0.996849	0.999531
12	0.543287	0.901234	0.993707	0.999063
13	0.295161	0.812223	0.987454	0.998126
14	0.087120	0.659706	0.975066	0.996255
15	0.007590	0.435211	0.950754	0.992525
16	0.000058	0.189409	0.903933	0.985106

in Proc. 30th Allerton Conf on Comm, Control, and computing oct 1992

Incorporating Dependent and Multimode Failures into Reliability Evaluation of Extra Stage Shuffle-Exchange MINs

Jerry L. Trahan, Daniel X. Wang, and Suresh Rai
Department of Electrical and Computer Engineering
Louisiana State University, Baton Rouge, LA 70803

Abstract

Multistage Interconnection Networks (MINs) provide a good communication medium between multiple processors and memory modules. Previous reliability evaluation efforts for MINs assumed that all failures are statistically independent and that no degraded operational modes exist, though these assumptions are inconsistent with realistic conditions. In this paper, we relax both assumptions and provide efficient algorithms for terminal, broadcast, and K-terminal reliability evaluation in the Shuffle-Exchange Network with an extra stage (SENE), a redundant path MIN. The shock model is used in a modified form to incorporate failure dependency and multiple operational modes into the reliability evaluation. For K-terminal reliability, let $k = |K|$. For an $N \times N$ SENE, the algorithms run in time $O(\log N)$, $O(\log N)$, and $O(k \log N)$, respectively.

1. Introduction

Parallel computers have developed very rapidly in response to the need for high speed computing in many applications. Several parallel computers, such as IBM's RP3 (Hsu *et al.*, 1987; Wang *et al.*, 1989), the University of Illinois' Cedar (Konicek *et al.*, 1991), and Purdue University's PASM (Schwederski *et al.*, 1991), employ a multistage interconnection network (MIN) to provide a communication medium between processors and processors or shared memory modules. A MIN consists of N inputs and N outputs and (typically) n stages of switching elements (SEs), where $n = \log_2 N$. An SE is generally a 2×2 crossbar network and provides either a straight (T-mode) or cross (X-mode) connection. The SEs in one stage are connected to the SEs in adjacent stages by links that are arranged in patterns. Based on these patterns, various MINs are called as Omega, Flip, Indirect binary cube, Modified data manipulator, baseline, and reverse baseline networks. All these networks are topologically equivalent (Bermond *et al.*, 1989). To improve the fault tolerance of the Omega network, an extra stage may be added to provide a redundant path from each input to each output. Thus, if one path fails due to faulty links and/or SEs, an input can still reach an output through another path. This type of Omega network is called the *Shuffle Exchange Network with an Extra stage* (*SENE*).

This work was supported in part by the Air Force Office of Scientific Research under grant AFOSR-91-0025.

K-terminal reliability is defined as the probability that a set of working paths exists from one specified input to each of a specified set K of outputs. For an $N \times N$ MIN and $|K| = 1$ (N), it is also known as *terminal (broadcast) reliability*.

Previous reliability evaluation algorithms for MINs assumed 2-mode fault models (working or failed) for each component and statistically independent failures (Botting *et al.*, 1989; Varma and Raghavendra, 1989; Cheng and Ibe, 1992; Trahan and Rai, 1992). These assumptions are common in reliability evaluation, as the evaluation problems are intractable for general networks (Colbourn, 1987). These assumptions, however, fail to adequately model real-world situations. For example, Davis *et al.* (1985) and Schwederski *et al.* (1991) discussed instances of dependent failures, or fault side-effects, in the PASM. The 2-mode model leads to an underestimate of the reliability because it does not allow a degraded operational mode of SEs. The assumption of independent failures leads to an overestimate of reliability. Recently, some researchers have addressed the problem of incorporating dependent failure into reliability computations for general networks (Boyles and Samaniego, 1984; Lam and Li, 1986; Le and Li, 1989).

In this paper, we present efficient algorithms for terminal, broadcast and K-terminal reliability evaluation of an SENE composed of identical SEs, allowing multimode and dependent failures. The algorithms assume a 4-mode model of an SE: a fully operational mode, two degraded operational modes, namely stuck-at-T mode and the stuck-at-X mode, and a completely failed mode. Moreover, we assume that links are reliable. We modify the shock model of Boyles and Samaniego (1984) to incorporate dependent failures into the reliability evaluation algorithms of Trahan and Rai (1992). For K-terminal reliability, let $k = |K|$. For an $N \times N$ SENE, the algorithms run in time $O(\log N)$, $O(\log N)$, and $O(k \log N)$, respectively.

The layout of the paper is as follows. Section 2 describes MIN basics. Section 3 discusses the shock model and explains its application to SENE reliability analysis. Section 4 presents results of the algorithms for terminal, broadcast, and K-terminal reliability of SENE and outlines the techniques used to develop them. For brevity, we present the results only. The derivations and proofs are discussed in (Wang, 1992).

2. Background

The Shuffle-Exchange Network with an Extra stage (SENE) (see figure in Cheng and Ibe (1992)) with N inputs and N outputs is defined to be a MIN with $\log_2 N+1$ stages of 2×2 SEs. The outputs of SEs in stage i connect to the inputs of SEs in stage $i+1$ by a shuffle connection, for $i = 0, 1, \ldots, n-1$. Let $SE_{i,j}$ denote the jth SE in stage i, where $0 \le i \le n$ and $0 \le j \le N/2-1$. To help develop a fault model, we consider an SE to be composed of two input controllers (ICs), two output controllers (OCs), and a central controller (CC). Figure 1 illustrates the structure of an SE. Paths through the network are established by a path request mechanism in which a CC decides whether the IC should transfer the input to the OC in T- or

X- connection mode.

3. Shock Model and Dependency Analysis

To analyze SE failure dependency, we use a shock model (Boyles and Samaniego, 1984). The shock model and the event based reliability model (EBRM) proposed by Lam and Li (1986) are identical. The shock model was defined for 2-mode components. We generalize its application to include multimode SEs in our reliability analysis. An extension of the EBRM to multimode components, the cause based multimode model (CBMM), does exist (Le and Li, 1989). Our model, however, is more simply defined as an extension of the shock model or EBRM than as a restricted application of the CBMM. Further, we can explain our generalized model in terms of shocks to subcomponents of SEs. Section 3.1 explains the concepts of the model while its application to SENE analysis is described in Section 3.2.

3.1. SHOCK MODEL

The shock model assumes that statistically independent *shocks*, which occur with known probability, cause the failure of network components. When a shock occurs, it causes the failure of a specific component or set of components. We say that the component or components are *affected* by the shock. A shock affecting a single component is called an *individual shock* (IS), while a shock affecting multiple components is called an *external shock* (ES).

For a network with n components, up to 2^n-1 shocks may be defined theoretically. Most shocks, however, may never happen in the real world. Therefore, we need to consider only those shocks whose occurrence is reasonable in the specified real world conditions.

3.2. SENE FAILURE ANALYSIS

To apply the shock model to a SENE in which SEs may operate in degraded modes, we will expand the notion of an IS, while leaving the notion of an ES unchanged. Instead of associating a single IS with a single component, we will associate one IS for each failed or degraded working mode of a component. Technically, a CC having a stuck-at logic fault can produce a degraded working mode for an SE. Class 1 shocks (defined below) model this scenario.

For ESs, we restrict our analysis to two modes only, so the occurrence of an ES causes all affected SEs to fail. SEs that are far apart are unlikely to be affected by a single ES. In general, the classes of ESs that we define are motivated by the failure of one SE causing the failure of other SEs due to the links connecting them. For shared-memory computers, each processor reads from or writes to the shared-memory through the MINs and so communication flows in both directions. When the forward (reverse) part of an SE is failed, this SE will send garbage messages to either one or both of the SEs in the next (previous) stage that are connected to it and may cause one or both of them to fail. Hence, we assume that an external

shock causes a failure in adjacent SEs either in the forward direction (towards the network outputs) or in the backward direction (from fault to network inputs). In a practical design, a failure of an output (input) controller can create a forward (backward) external shock. Such dependent failures have been noted in MINs by Davis *et al.* (1985) and Schwederski *et al.* (1991). In the terminology of Schwederski *et al.*, the shocks that we have described above correspond to fault side-effects with forward reach of 1, backward reach of 1, and span of 2. Class 2, 3out, and 3in shocks (defined below) model this scenario.

To help compute the reliabilities, we consider the following *classes* of shocks. Each class of shock will affect a certain structured set of SEs. For example, a Class 3out shock (defined below) will affect an SE and the two SEs to which it is connected in the next stage. We define such a shock for every SE. Because the structures affected by the same class of shocks are the same and all SEs are identical, the probabilities that the same class of shock occur are identical for all such shocks. The probability that each class of shock occurs may be time dependent. The purpose of our work is to find out a relationship between the probabilities of each class of shock and the reliability of the whole network.

Class 1 shock. Exactly one SE is damaged or fails.

A Class 1 shock is an IS that affects only one SE. We modify the shock model as described above to handle stuck-at-T and stuck-at-X modes of SEs. Let $Z_{ij}(1,f)$ denote the shock that affects SE_{ij} to be completely failed, let $Z_{ij}(1,t)$ denote the shock that affects SE_{ij} to be stuck at T mode, and let $Z_{ij}(1,x)$ denote the shock that affects SE_{ij} to be stuck at X mode. The probabilities that the three Class 1 shocks occur are p_f, p_t, and p_x, respectively. Let p_w denote the probability that the three Class 1 shocks affecting SE_{ij} do not occur, hence, $p_w = 1-(p_f+p_t+p_x)$.

Class 2 shock. Exactly two SEs fail simultaneously.

A Class 2 shock is an ES that affects two SEs connected by a link. Four Class 2 shocks affect each SE SE_{ij}, if SE_{ij} is not in the input or output stages. These four shocks will be denoted as $Z_{ij}(2,k)$, where k = 1, 2, 3, 4 (Figure 2(a)). If SE_{ij} is in the input stage, then only $Z_{ij}(2,3)$ and $Z_{ij}(2,4)$ affect it. If SE_{ij} is in the output stage, then only $Z_{ij}(2,1)$ and $Z_{ij}(2,2)$ affect it. Let the probability that a Class 2 shock occurs be p_2 and does not occur be $q_2 = 1-p_2$.

Class 3out shock and Class 3in shock. Exactly three SEs fail simultaneously.

Class 3out shock.

A Class 3out shock is an ES that affects an SE and the two SEs to which it is connected by its output links. Each SE in a SENE is affected by three Class 3out shocks denoted as $Z_{ij}(3,k)$, where k = 1, 2, 3, except for SEs in the input and output stages (Figure 2(b)). If SE_{ij} is in the input stage, then only $Z_{ij}(3,3)$ affects it. If SE_{ij} is in the output stage, then only $Z_{ij}(3,1)$ and $Z_{ij}(3,2)$ affect it. Let the probability that a Class 3out shock occurs be p_{3o} and does not occur be $q_{3o}= 1-p_{3o}$.

Class 3in shock.

A Class 3in shock is an ES that affects an SE and the two SEs to which it is connected by its input links. There are obviously three Class 3in shocks affecting each SE, denoted as $Z_{ij}(3,k)$, where $k = 4, 5, 6$, except for SEs in the input and output stages (Figure 2(c)). SE_{ij} is affected by only $Z_{ij}(3,5)$ and $Z_{ij}(3,6)$ if it is in the input stage and is affected by only $Z_{ij}(3,4)$ if it is in the output stage. Let the probability that a Class 3in shock is up be p_{3i} and down be $q_{3i} = 1-p_{3i}$.

4. Reliability Evaluation

We now present the results for terminal (TR), broadcast (BR), and K-terminal (KR) reliability measures. Trahan and Rai (1992) developed a straightforward algorithm for TR and recursive algorithms for BR and KR of a SENE under assumptions of independent and 2-mode SE failures. They noted that SENE paths form a simple series-parallel graph for TR and a pair of intersecting binary trees for BR and KR. To incorporate dependent and multimode failures, we follow their concept, but must include a careful and much more detailed accounting of the shocks that may affect the SEs on the paths. Note that a single ES may affect one, two, or three SEs on the relevant paths.

To compute TR, note the following.

a. For a SENE, there exist two paths from each input to each output. The two paths share an SE in the input stage and an SE in the output stage, but are otherwise disjoint.

b. A routing tag can be used to set the connections in switches on a path from an input s to an output d. If an input number is $s = s_1 s_2 \ldots s_n$ and an output number is $d = d_1 d_2 \ldots d_n$, where $s_1, s_2, \ldots, s_n$ and $d_1, d_2, \ldots, d_n$ are the bits of the binary representation of s and d respectively, then the routing tag from s to d is $r = r_1 r_2 \ldots r_n$, where $r_1, r_2, \ldots, r_n$ are given by $r_i = s_i \oplus d_i$, for each $1 \le i \le n$.

Theorem 1. For an input s and an output d in an $N \times N$ SENE, the terminal reliability can be computed using the equation below in $O(\log N)$ time.

$$TR(N,s,d) = (p_t + p_w)^{n+1-\sum_{i=1}^{n} r_i} (p_x + p_w)^{\sum_{i=1}^{n} r_i} q_2^{3n} (q_{3o} q_{3i})^{2n}$$
$$+ (p_t + p_w)^{n-1+r_n-\sum_{i=1}^{n-1} r_i} (p_x + p_w)^{2-r_n+\sum_{i=1}^{n-1} r_i} q_2^{3n} (q_{3o} q_{3i})^{2n}$$
$$- p_w^2 \left[(p_t + p_w)^{n-1-\sum_{i=1}^{n-1} r_i} (p_x + p_w)^{\sum_{i=1}^{n-1} r_i} \right]^2 q_2^{6n-4} (q_{3o} q_{3i})^{4(n-1)}$$

Based on the structure of the broadcast paths in the SENE (Figure 3), BR can be computed by a recurrence algorithm. This algorithm is described by Theorem 2.

Theorem 2. For an input s in an $N\times N$ SENE, the broadcast reliability can be evaluated using the equation below in $O(\log N)$ time.

$$BR(N) = ((p_t + p_x)p_w^{-1} - 2p_{3o}q_{3o}^{-1})q_2^{-1}\beta^N + \beta q_{3i}q_{3o}^{-1}BR'(N),$$

where $\alpha = p_w(q_2)^3(q_{3o}q_{3i})^2$, $\beta = p_w q_2^2 q_{3o}q_{3i}$, and $BR'(N)$ can be computed by

$$BR'(N) = \alpha^2 q_{3o}^{-2}[BR'(N/2)]^2 + 2(p_t + p_x)p_w^{-1}\alpha\beta^{N/2}BR'(N/2) + 2p_w^{-2}q_{3o}[p_w(p_f + \alpha^{-1}p_w - 1) + p_t p_x q_{3o}]\beta^N.$$

The base case $BR'(4)$ is defined as

$$BR'(4) = [(1-p_f)^2 + 2(p_t p_x + p_t + p_x - p_w)(1-p_f) + 2p_w(q_2 q_{3o})^{-1}]p_w^2 q_2^6 q_{3o}^4 q_{3i}^2.$$

Definition. For K-terminal reliability, we describe each SE on a path from a specified input s to an output in K as *marked*.

Theorem 3. For an input s and a set K of outputs in an $N\times N$ SENE, where $k = |K|$, the K-terminal reliability can be computed by the equation below in $O(k \log N)$ time.

$$KR(K,N,s) = \left[(p_t + p_x - 2p_w p_{3o}q_{3o}^{-1})KR_1(K,N,s) + p_w q_{3o}^{-1}KR_2(K,N,s)\right]q_2^2 q_{3o}q_{3i}^2.$$

Values for $KR_1(K,N,s)$ and $KR_2(K,N,s)$ can be computed by recurrence expressions according to the different cases enumerated below. The base case for N=4 can be computed by considering several different cases. For the sake of brevity, the results for the base case are not enumerated.

Case I: Only one child of considered SEs is marked.

1. The T mode in these SEs allow working paths from input s to the k outputs, then

$KR_1(K,N,s)$ and $KR_2(K,N,s)$ can be computed by

$$KR_1(K,N,s) = (p_t + p_w)q_2^3(q_{3o}q_{3i})^2 KR_1\left(K,\frac{N}{2},s\right)$$

$$KR_2(K,N,s) = \left[(p_t + p_w)^2 KR_2\left(K,\frac{N}{2},s\right) + 2(p_t + p_w)(p_x + p_f + \alpha^{-1}p_w - 1)KR_1\left(K,\frac{N}{2},s\right)\right] \cdot q_2^6(q_{3o}q_{3i})^4.$$

2. The X mode in these SEs allow working paths from input s to the k outputs, then

$KR_1(K,N,s)$ and $KR_2(K,N,s)$ can be computed by the equations above, exchanging p_t and p_x.

Case II: Both children of considered SEs are marked. Let K_l and K_r be the subsets of K that lie in the left and right subgraphs, respectively.

$KR_1(K,N,s)$ can be computed by

$$KR_1(K,N,s) = KR_1\left(K_l,\frac{N}{2},s\right)KR_1\left(K_r,\frac{N}{2},s\right) \cdot p_w q_2^3 q_{3o}q_{3i}^2.$$

$KR_2(K,N,s)$ can be computed by

1. If input s can access the right (left) subgraph if considered SEs are set in T (X) mode, then

$$KR_2(K,N,s) = \Big[p_w^2 q_{3o}^{-2} KR_2\Big(K_l,\frac{N}{2},s\Big)KR_2\Big(K_r,\frac{N}{2},s\Big)$$
$$+ 2p_l p_w q_{3o}^{-1} KR_1\Big(K_l,\frac{N}{2},s\Big)KR_2\Big(K_r,\frac{N}{2},s\Big)$$
$$+ 2p_x p_w q_{3o}^{-1} KR_2\Big(K_l,\frac{N}{2},s\Big)KR_1\Big(K_r,\frac{N}{2},s\Big)$$
$$+2\Big(p_w\big(p_f + \alpha^{-1}p_w - 1\big)q_{3o}^{-1} + p_l p_x\Big)KR_1\Big(K_l,\frac{N}{2},s\Big)KR_1\Big(K_r,\frac{N}{2},s\Big)\Big]$$
$$\cdot q_2^6 q_{3o}^4 q_{3i}^4.$$

2. If input s can access the left (right) subgraph if considered SEs are set in T (X) mode, then KR_2 can be computed by the equation above, exchanging p_l and p_x.

References

J. C. Bermond, J. M. Fourneau, and A. Jean-Marie (1989), "A graph theoretical approach to equivalence of multistage interconnection networks," *Discr. Appl. Math.*, vol. 22, pp. 201-214.

C. Botting, S. Rai and D. P. Agrawal (1989), "Reliability computation of multistage interconnection networks," *IEEE Trans. Reliability*, vol. 38, no. 1, pp. 138-145.

R. A. Boyles and F. J. Samaniego (1984), "Modeling and inference for multivariate binary data with positive dependence," *J. Am Stat. Assoc.*, vol. 79, pp. 188-193.

X. Cheng and O. C. Ibe (1992), "Reliability of a class of multistage interconnection networks," *IEEE Trans. Par. Distr. Comput.*, vol. 3, no. 2, pp. 241-246.

C. Colbourn (1987), *The Combinatorics of Network Reliability*, Oxford Univ. Press, New York.

N. J. Davis IV, W. T.-Y. Hsu, and H. J. Siegel (1985), "Fault location techniques for distributed control interconnection networks," *IEEE Trans. Comput.*, vol. C-34, pp. 902-910.

Y. Hsu, W. Kleinfelder, and C. J. Tan (1987), "Design of a combining network for the RP3 project," *Proc. of Int'l Conf. on VLSI Technology, Systems, and Applications*, pp. 349-353.

J. Konicek *et al.* (1991), "The organization of the CEDAR system," *Proc. Int'l Conf. on Parallel Processing*, pp. I_49-I_56.

Y. F. Lam and V. O. K. Li (1986), "Reliability modeling and analysis of communication networks with dependent failures," *IEEE Trans. Commun.*, vol. COM-34, no. 1, pp. 82-84.

K. V. Le and V. O. K. Li (1989), "Modeling and analysis of systems with multimode components and dependent failures," *IEEE Trans. Reliability*, vol. 38, no. 1, pp. 68-75.

T. Schwederski, E. Bernath, G. Roos, W. G. Nation, and H. J. Siegel (1991), "Fault side-effects in fault tolerant multistage interconnection networks," *Proc. of Int'l Conf. on Parallel Processing*, pp. I_313-I_317.

J.L. Trahan and S. Rai (1992), "Reliability evaluation of extra stage shuffle-exchange MINs," to appear in *Proc. 1992 Int'l. Conf. Computer Communication*, Genova, Italy.

A. Varma and C. S. Raghavendra (1989), "Reliability analysis of redundant-path interconnection networks," *IEEE Trans. Reliability*, vol. 38, no. 1, pp. 130-137.

D. X. Wang (1992), "Incorporating dependent and multimode failures into reliability evaluation of multistage interconnection networks," M.S. thesis, Louisiana State Univ.

S. Wang, Y. Hsu and C.J. Tan (1989), "A high performance VLSI message switch for multi-processor systems," Research Report No. 54874, IBM Research Division.

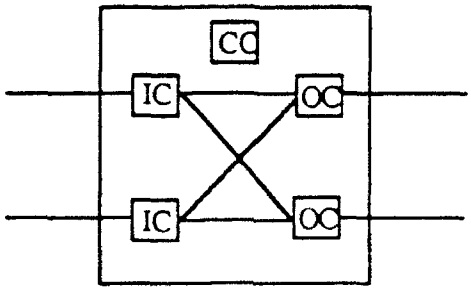


Figure 1. Structure of an SE.
IC: input controller, OC: output controller
CC: central controller

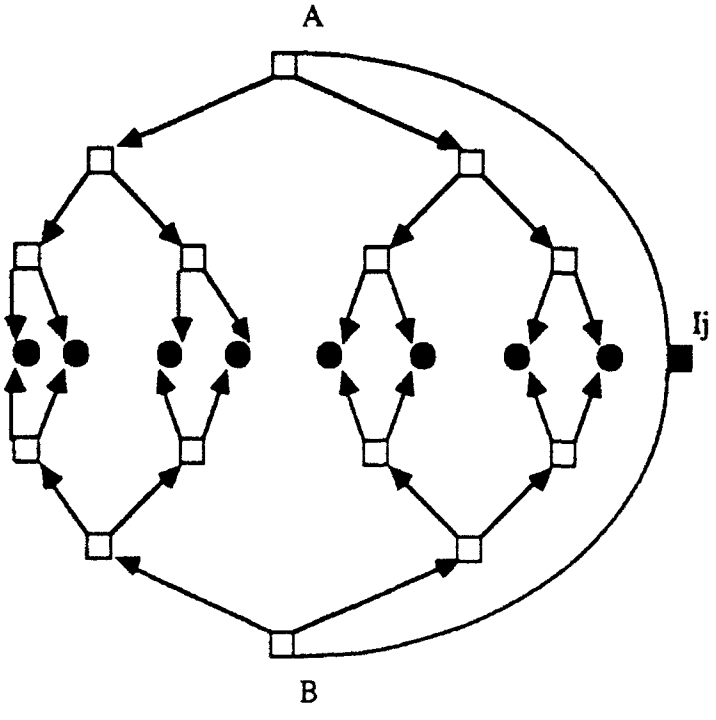
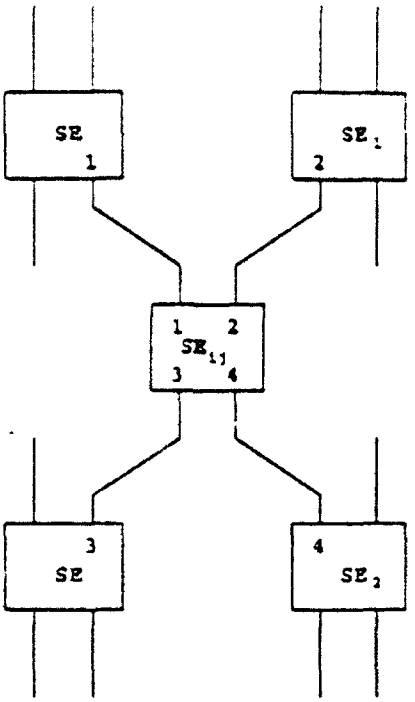
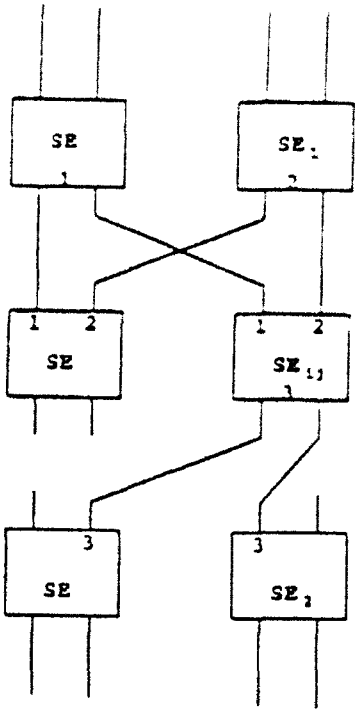


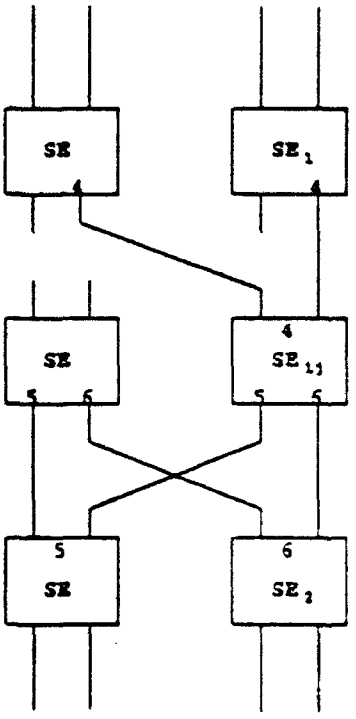
Figure 3. Broadcast paths in SENE
● denotes an output stage SE



(a) Class 2 shock



(b) Class 3out shock



stage i–1

stage i

stage i+1

(c) Class 3in shock

Figure 2. Classes of shocks

in Proc 1992 Int Conf on Computer [illegible]
Geneva, Italy, Oct 1992 1

Reliability Evaluation in Extra Stage Shuffle-Exchange MINs

Jerry L. Trahan and Suresh Rai

Dept. of Electrical and Computer Engineering
Louisiana State University, Baton Rouge, LA 70803

phone: (504) 388-4830
fax: (504) 388-5700
e-mail: trahan@max.ee.lsu.edu

Thu, Mar 5, 1992

Summary and Conclusion. Multistage interconnection networks (MINs) are a widely studied means of interconnecting processors to memory or processors to processors by stages of switches. In this paper, we have presented a set of efficient reliability evaluation algorithms for the terminal reliability, broadcast reliability, and K-terminal reliability problems in shuffle-exchange network with an extra stage (SENE) MIN with time complexities $O(\log\log N)$, $O(\log N)$, and $O(N \log N)$ time, respectively. For each of these problems, the best previous approaches took time exponential in N.

This work was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-91-0025.

1. Introduction

To achieve faster computing speeds imperative for many computer applications, the use of multiple processors operating in parallel is necessary. Consequently, the reliability of the network interconnecting these processors is of notable importance, as is the ability to quickly evaluate whether the network can implement a desired set of connections. The problem of exact reliability evaluation, however, is computationally intractable for most reliability measures in general networks. In particular, the problems of terminal reliability, broadcast reliability, and K-terminal reliability evaluation are $\#P$-complete (Ball, 1986). K-terminal reliability is the probability that a path exists from one specified node to each of a specified set K of nodes; broadcast (terminal) reliability is a special case of K-terminal reliability in which K is the set of all (one) output nodes (node). For some restricted cases, though, a network offers sufficient structure that the reliability may be efficiently evaluated (Colbourn, 1987).

Multistage interconnection networks (MINs) are a widely studied means of interconnecting processors to memory or processors to processors by stages of switches. MINs are also increasingly used in experimental systems. MINs are an integral part of the design of such large scale projects as PUMPS, CEDAR, and PASM (Siegel, 1985). Therefore, the problem of reliability evaluation of MINs is of interest. In this paper, we present simple and efficient algorithms for terminal, broadcast, and K-terminal reliability evaluation of the shuffle-exchange network with an extra stage. For the reliability problems, we utilize a stochastic model of the MIN in which SEs may fail with a known probability and links are always working. The terminal and broadcast reliability evaluation algorithms run within time $O(\text{loglog } N)$ and $O(\log N)$ for a network of size N, respectively. The K-terminal reliability evaluation algorithm runs within time $O(N \log N)$. Varma and Raghavendra (1989) obtained similar efficient algorithms for terminal reliability and broadcast reliability evaluation of the Generalized INDRA network, Merged Delta network, and Augmented C-network.

The structure of the paper is as follows. In Section 2, we present definitions and describe some background results used throughout the paper. Section 3 describes the reliability evaluation algorithms and analyses their complexity issues.

2. Definitions and Background

Multistage interconnection networks essentially comprise *switching elements* (SEs) and links between switching elements. MINs may contain many combinations of switch sizes (Feng, 1981). For our discussion, we restrict ourselves to shuffle-exchange MINs built from 2-input, 2-output SEs. Such a shuffle-exchange MIN has $N = 2^n$ inputs and outputs and n stages, with each stage comprising $N/2$ switches. The stages are numbered from 1 to n. The outputs of SEs in stage i connect to the inputs of stage $i+1$ by a shuffle connection, for $1 \le i < n$. A SENE has $n+1$ stages, numbered 0 to n. SEs at the input, stage 0, are labeled $I_0, I_1, \ldots, I_{N/2-1}$. SEs at the output stage are labeled $O_0, O_1, \ldots, O_{N/2-1}$. SEs at stage i are labeled $(i-1)N/2, (i-1)N/2 + 1, \ldots, iN/2 - 1$, for $1 \le i < n$. For the sake of discussion, assume that the MIN connects N input processors to N output processors.

For any pair of input and output processors, the SENE possesses exactly two paths from the input to the output. These paths share an SE in the input stage and an SE in the output stage, but are otherwise disjoint. We will designate the path through the smaller numbered SEs as the *upper path* and the other as the *lower path*. For example, the upper path from input 0 to output 0 contains SEs I_0, 0, 8, 16, and O_0, while the lower path contains SEs I_0, 1, 10, 20, and O_0 (Figure 1).

In a SENE, each SE in stage 0, the input stage, is connected to a pair of SEs in stage 1. Each of these SEs is the root of a complete binary tree of SEs of height n whose leaves are the SEs of stage n, the output stage. The two trees are disjoint, except that the leaves of the trees are identical. (See Figure 1.) We call the tree (including the leaves) rooted at the smaller numbered switch as the *upper broadcast tree* (BT_U) and the tree (including the leaves) rooted at the larger numbered switch as the *lower broadcast tree* (BT_L). Omitting the input stage SEs, the set of upper paths from any input processor forms the upper broadcast tree, and the set of lower paths forms the lower broadcast tree.

We define the *upper network* as the set of upper paths from each input to each output, omitting the input stage SEs, and we define the *lower network* as the set of lower paths from each input to each output, omitting the input stage SEs. Each input stage SE is connected to one SE in stage 1 of the upper network and one SE in stage 1 of the lower network. Figure 1 depicts the upper and lower networks of a 16×16 SENE, noting the connections of the input stage SEs and depicting the output stage SEs in the center. These very regular paths from an input to the outputs offers us the structure necessary to efficiently solve the reliability and decision problems.

Note from Figure 1 that the upper and lower networks are symmetrical. For a given SE g in stage i of the upper network, there is an SE g' in the lower network in the corresponding position in the same stage. Let $L(g)$ denote this SE. For example, in a 16×16 SENE, $L(0) = 1$ and $L(17) = 21$. For SE h in the lower network, let $U(h)$ denote the SE in the corresponding position in the upper network, so $U(1) = 0$ and $U(21) = 17$ in our example.

Because of the regular connection pattern between stages of SEs, it is straightforward to list the SEs on a path from a specified input to a specified output or, to determine the SEs connected to a given SE j. We refer to an SE k in a stage preceding that of SE j such that a path exists in the fault-free SENE from SE k to SE j as an *ancestor* of SE j. We refer to an SE k in a stage following that of SE j such that a path exists in the fault-free SENE from SE j to SE k as a *descendant* of SE j.

We assume that the processor running our algorithm is a Random Access Machine that can compute addition, subtraction, multiplication, division, and bitwise Boolean operations in one unit of time (Trahan *et al.*, 1991).

For an integer s, let $\#s$ denote its representation in binary.

Lemma 2.1. For any SE g in an N×N SENE, the following can be computed in constant time, given a table T that can be generated in $O(\log N)$ time:

(i) the stage i in which SE g is located,

(ii) the pair of SEs in stage i+1 to which the outputs of SE g are connected,

(iii) the pair of SEs in stage i–1 connected to the inputs of SE g,

(iv) for an SE h in a stage preceding stage i, whether SE h is an ancestor of SE j,

(v) for an SE h in a stage following stage i, whether SE h is a descendant of SE j,

(vi) whether SE g is in the upper or lower network, and

(vii) if SE g is in the upper (lower) network, the SE $L(g)$ ($U(g)$) in the corresponding position in the lower (upper) network.

Lemma 2.2. For any given input I_j and output O_k in an $N \times N$ SENE, the upper and lower paths from I_j to O_k can be generated in $O(\log N)$ time.

Terminal reliability (*TR*) is the probability that at least one path exists from a given input processor to a given output processor of the network. *K-terminal reliability* is the probability that at least one set of paths exists from a given input processor to each processor in a set K of output processors of the network. *Broadcast reliability* (*BR*) is the probability that at least one set of paths exists from a given input processor to each output processor of the network. BR is a special case of K-terminal reliability. *Network reliability* is the probability that at least one set of paths exists to connect each input processor to each output processor.

In the SENE, each input processor is connected to the input of a single SE in the input stage, and each output processor is connected to the output of a single SE in the output stage. In the following, we describe our algorithms based on the input (output) stage SE to which an input (output) processor is connected, rather than based on the input (output) processor itself.

We make the following assumptions for the reliability problems. Each SE in the input stage and output stage is always working. Each SE in the other (intermediate) stages is working with probability p and failed with probability $q = 1 - p$. Individual SE failure probabilities are statistically independent. Each link is always working.

We will later show how to relax the assumption that the failure probability of each SE is identical.

3. Reliability Evaluation Algorithms

3.1. TERMINAL RELIABILITY

Given the structure of the SENE mentioned above, the TR problem is easily solved. Let I_j be the specified input and O_k be the specified output. An $N\times N$ SENE contains exactly two disjoint paths, that is, the upper and lower paths, each of length $n = \log_2 N$, from I_j to O_k. (Note: All logarithms are taken to base 2 in this paper.) The graph is simply series-parallel. Let TR(N) denote the terminal reliability of an $N\times N$ SENE.

$$\mathrm{TR}(N) = 1 - (1 - p^{\log N - 1})^2.$$

Theorem 3.1. The terminal reliability of an $N\times N$ SENE can be evaluated in $O(\log\log N)$ time.

3.2. BROADCAST RELIABILITY

The structure of the SENE allows us to use a recursive approach to evaluating the broadcast reliability of the SENE. For the BR problem, let I_j be the specified input SE, let A be the root of $\mathrm{BT_U}$, let C and E be the two SEs to which A is connected in stage 2, let B be the root of $\mathrm{BT_L}$, and let D and F be the two SEs to which B is connected in stage 2, where the label of SE C is less than that of E and the label of D is less than that of F. (See Figure 2.)

Input I_j may reach all outputs by paths through A only, through B only, or some outputs through A and the rest through B. We handle each of these cases separately. Case 1 comprises instances in which A is working and B is failed; Case 2 comprises instances in which A is failed and B is working; and Case 3 comprises instances in which both A and B are working. Each case describes a disjoint collection of instances, so the overall reliability will be the sum over the three cases of the probability that a set of paths exists from I_j to each output in each case.

Case 1. SE A is working and SE B is failed. No paths exist from I_j to any inner node in $\mathrm{BT_L}$, so the graph of nodes reachable from I_j comprises I_j, an edge from I_j to A, and a complete binary tree of height n rooted at A. Exactly one path exists from I_j to each output, so every node in the binary tree

must be working. This tree contains $N/2$ leaves and $N/2 - 1$ inner nodes. The reliability of this case is as follows.

$$R_{A\,\bar{B}}(N) = p^{N/2-1}q.$$

Case 2. SE A is failed and SE B is working. The analysis is analogous to that of Case 1.

$$R_{\bar{A}B}(N) = p^{N/2-1}q.$$

Case 3. Both SE A and SE B are working. Working paths exist from I_j to SEs C, D, E, and F. Call the outputs O_0 through $O_{N/4-1}$ the left half outputs, and call the outputs $O_{N/4}$ through $O_{N/2-1}$ the right half outputs. Input I_j can reach the left half outputs through SEs C and D, and I_j can reach the right half outputs through SEs E and F. Observe that the probability P_L that I_j can reach all the left half outputs is equal to the probability P_R that I_j can reach all the right half outputs. The probability that I_j can reach all the outputs is equal to $P_L P_R = (P_L)^2$. Evaluating P_L reduces to the same broadcast reliability evaluation problem in a network with half the number of outputs.

$$R_{AB}(N) = p^2(BR(N/2))^2.$$

Putting the three cases together, we obtain the following recurrence for a SENE with N outputs.

$$BR(N) = 2p^{N/2-1}q + p^2(BR(N/2))^2.$$

The base case for the recurrence is $BR(2) = 2pq + p^2$.

Theorem 3.2. The broadcast reliability of an $N \times N$ SENE can be evaluated in $O(\log N)$ time.

Proof. We precompute $p^{n2^{-i}-1}$ for $i = 1, 2, \ldots, \log N$ in time $O(\log N)$. We then evaluate the recurrence equation in a constant amount of time for each of $\log N$ levels of recursion and, so evaluate the broadcast reliability of an $N \times N$ SENE in time $O(\log N)$. ∎

Time $O(\log N)$ to evaluate BR is far better than the time complexity of previous algorithms using the sum of disjoint products method and running in time exponential in N (Botting *et al.*, 1989;

Rai and Trahan, 1989; Kulkarni and Trahan, 1991). Theorem 3.2 also establishes that the problem of evaluating the broadcast reliability for a SENE is not #P-complete, as is the case for a general network.

The recurrence obtained for BR evaluation is very similar to recurrences generated by Varma and Raghavendra (1989) for BR evaluation of other MINs. Their redundancy graphs for the Generalized Indra Network and Augmented C Network are very similar to the broadcast tree structure shown in Figure 2 for the SENE. Their recurrence for BR on the Augmented C Network is almost identical to that for the SENE.

3.3. DIFFERENT SWITCH RELIABILITIES

We now extend the solution method to two related problems. The first is the BR problem for the SENE if SEs are allowed different probabilities of working, and the second is the K-terminal reliability problem. The K-terminal reliability evaluation algorithm will use the BR evaluation algorithm for different switch reliabilities as a building block.

Suppose that the reliabilities of individual SEs may differ. Let p_i denote the probability the SE i is working. We will follow the previous decomposition approach with the same cases resulting, but will evaluate the contribution of each case to the total reliability differently. The reliability function BR′ now has two arguments, G, the graph and associated reliabilities, and N, the size of the SENE. Let G_L denote the decomposed graph containing the left half outputs and associated reliabilities, and let G_R denote the decomposed graph containing the right half outputs and associated reliabilities.

The complete recurrence, hence recursive algorithm, for this situation is as follows.

$$\mathrm{BR}'(G, N) = q_B \prod_{s \in BT_U} p_s + q_A \prod_{s \in BT_L} p_s + p_A p_B \ \mathrm{BR}'(G_L, N/2)\ \mathrm{BR}'(G_R, N/2).$$

The base case for the recurrence is $\mathrm{BR}'(G, 2) = p_A q_B + q_A p_B + P_A p_B$.

Theorem 3.3. The broadcast reliability of an $N \times N$ SENE in which each switch may have a different reliability can be evaluated in $O(N \log N)$ time.

3.4. K-TERMINAL RELIABILITY EVALUATION

K-terminal reliability is the probability that at least one set of paths exists from a given input processor to each output processor in a set K of size k. BR is a special case of K-terminal reliability in which set K comprises the set of all output processors; TR is a special case of K-terminal reliability in which set K comprises a single element. Given an $N \times N$ SENE, a specified input I_j, and a set $K = \{O_a, O_b, \ldots, O_c\}$, where $|K| = k$, we wish to evaluate the probability that I_j can reach each output $O_m \in K$. We present two algorithms for computing K-terminal reliability, the first for instances in which $k > \log N$, and the second for instances in which $k \leq \log N$. For these algorithms, the K-terminal reliability is computed the same way whether switch reliabilities are the same or different, so for clarity and generality, we describe the situation in which these reliabilities are different.

Both algorithms start with the same initialization procedure as follows. Set up an initially empty array B, where element $B(g)$ corresponds to SE g. Array B contains $O(N \log N)$ elements. For each $O_m \in K$ and each SE g in the upper path or in the lower path from I_j to O_m, mark element $B(g)$. Each path can be computed in $O(\log N)$ time by Lemma 2.2, and there are $2k$ paths to trace, so this initialization takes $O(k \log N)$ time.

Algorithm K1: $k > \log N$.

To evaluate the K-terminal reliability, execute the algorithm above for the BR problem with different switch reliabilities, making the following modification. If SE s is marked, then leave its reliability as p_s; if SE s is not marked, then treat its reliability as 1. This modification treats the parts of the broadcast trees from input I_j that reach only outputs not in K as being completely reliable, so the result returned by this algorithm is exactly the K-terminal reliability. The time complexity of this algorithm is $O(N \log N)$, as for the different switch reliabilities problem.

Algorithm K2: $k \leq \log N$.

If an SE g is not marked, then no path from I_j to any output in the set K contains SE g. Note further that no path from I_j to any output in the set K contains any descendant of SE g because of the tree structure of the broadcast paths. Therefore, unmarked SEs and their successors will be handled as whole subtrees without further recursion.

The same decomposition approach will again be followed, but the contribution of each case to the total reliability will be evaluated differently depending on whether an SE is marked or not. The reliability function *KR* has two arguments, *G*, the graph and associated reliabilities, and *N*, the size of the SENE. For the specified switches SE *A* and SE *B*, either both are marked or both are unmarked because they share the same set of outputs that are descendants in stage *n*. Let $BT_{U,m}$ ($BT_{L,m}$) denote the set of marked SEs in BT_U (BT_L).

The recurrence, and hence recursive Algorithm K2, is specified below.

$$KR(G, N) = q_B \prod_{s \in BT_{U,m}} p_s + q_A \prod_{s \in BT_{L,m}} p_s + p_A p_B \; KR(G_L, N/2)\, KR(G_R, N/2),$$ if both *A* and *B* are marked;

$KR(G, N) = 0$, if both *A* and *B* are unmarked.

The base cases for the recurrence is $KR(G, 2) = p_A q_B + q_A p_B + P_A p_B$, if both *A* and *B* are marked; $KR(G, 2) = 0$, if both *A* and *B* are unmarked.

The time to evaluate $KR(G, N)$ is the sum of the time to evaluate each of the three terms. The first two terms may be evaluated in $O(k \log N)$ time. The time to evaluate the third term is the sum of the times to evaluate two *KR* functions for graphs with $N/2$ outputs. Thus, the overall time to evaluate $KR(G, N)$ is as follows.

$$\begin{aligned} T_{KR}(N) &= 2ck \log N + 2T_{KR}(N/2), \text{ where } c \text{ is a constant} \\ &= ck \sum_{i=1}^{\log N} 2^i (\log N - (i-1)) \\ &= O(kN). \end{aligned}$$

This time measure is an overestimate, as it does not account for the fact that the size *k* of the set of outputs of interest decreases at lower levels of recursion. (The amount of decrease at each level of recursion depends on the exact set of elements in *K*.) Since the initialization time is $O(k \log N)$, the overall time to execute Algorithm K2 to compute *K*-terminal reliability for a SENE of size *N* is $O(kN)$.

Theorem 3.4. The K-terminal reliability for a SENE can be computed in $O(N \cdot \min\{k, \log N\})$ time.

References

M. O. Ball (1986), "Computational Complexity of Network Reliability Analysis: An Overview," *IEEE Trans. Reliability*, vol. R-35, no. 3, pp. 230-239.

C. Botting, S. Rai, and D. P. Agrawal (1989), "Reliability Computation of Multistage Interconnection Networks," *IEEE Trans. Reliability*, vol. 38, no. 1, pp. 138-145.

C. J. Colbourn (1987), *The Combinatorics of Network Reliability*, Oxford University Press, New York.

T. Feng (1981), "A Survey of Interconnection Networks," *Computer*, vol. 14, no. 12, pp. 12-27.

A. Kulkarni and J. L. Trahan (1991), "Broadcast Reliability Evaluation of Multistage Interconnection Networks," *Proc. Southeastcon '91*, Williamsburg, VA, pp. 432-436.

S. Rai and J. Trahan (1989), "Computing Network Reliability of Redundant MINs", *Proc. 21st Southeastern Symposium on System Theory*, Tallahassee, FL, pp. 221-225.

H. J. Siegel (1985), *Interconnection Networks for Large Scale Parallel Processing: Theory and Case Studies*, Lexington Books, Lexington, MA.

J. L. Trahan, M. C. Loui, and V. Ramachandran (1991), "Multiplication, Division, and Shift Instructions in Parallel Random Access Machines," to appear in *Theoretical Computer Science*.

A. Varma and C. S. Raghavendra (1989), "Reliability Analysis of Redundant-Path Interconnection Networks," *IEEE Trans. Reliability*, vol. 38, no. 1, pp. 130-137.

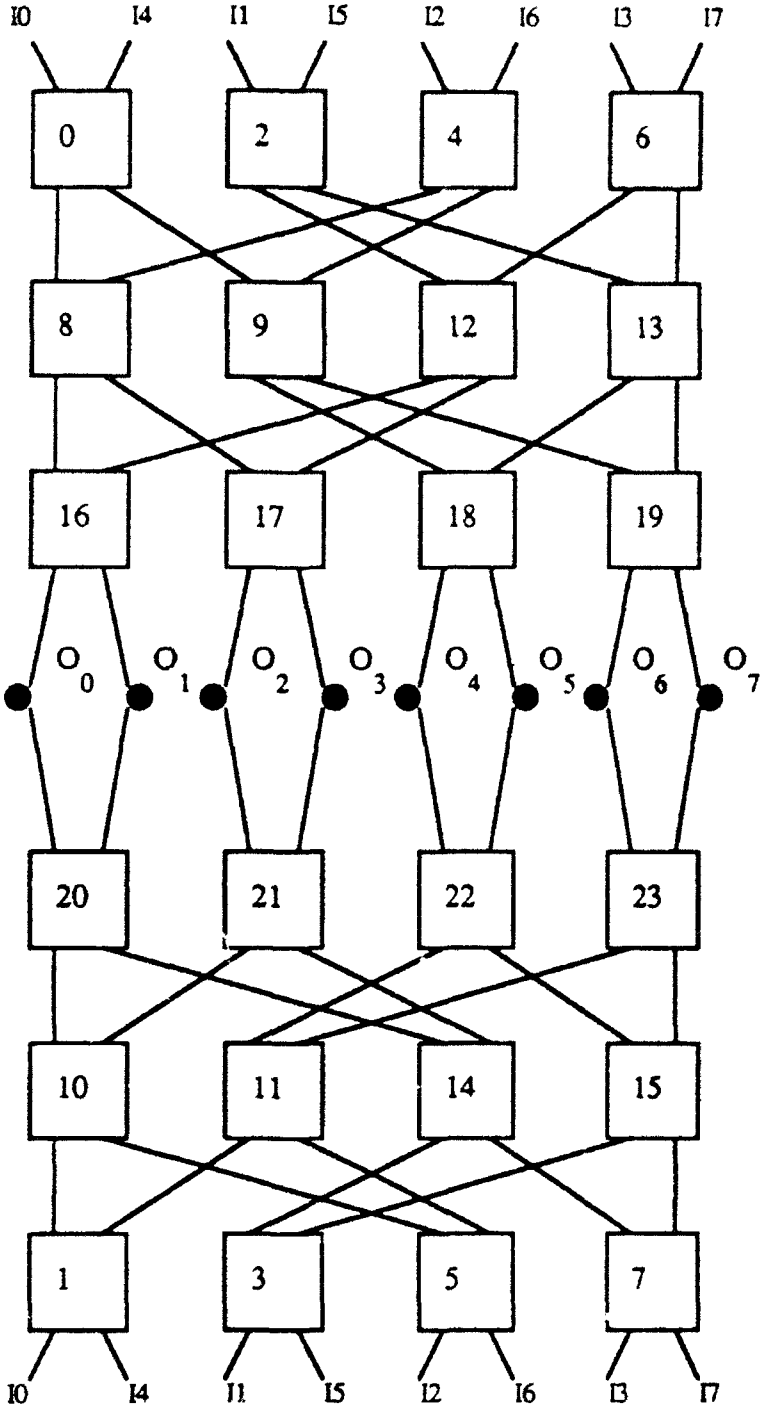


Figure 2. Upper and lower networks for a 16 x 16 SENE

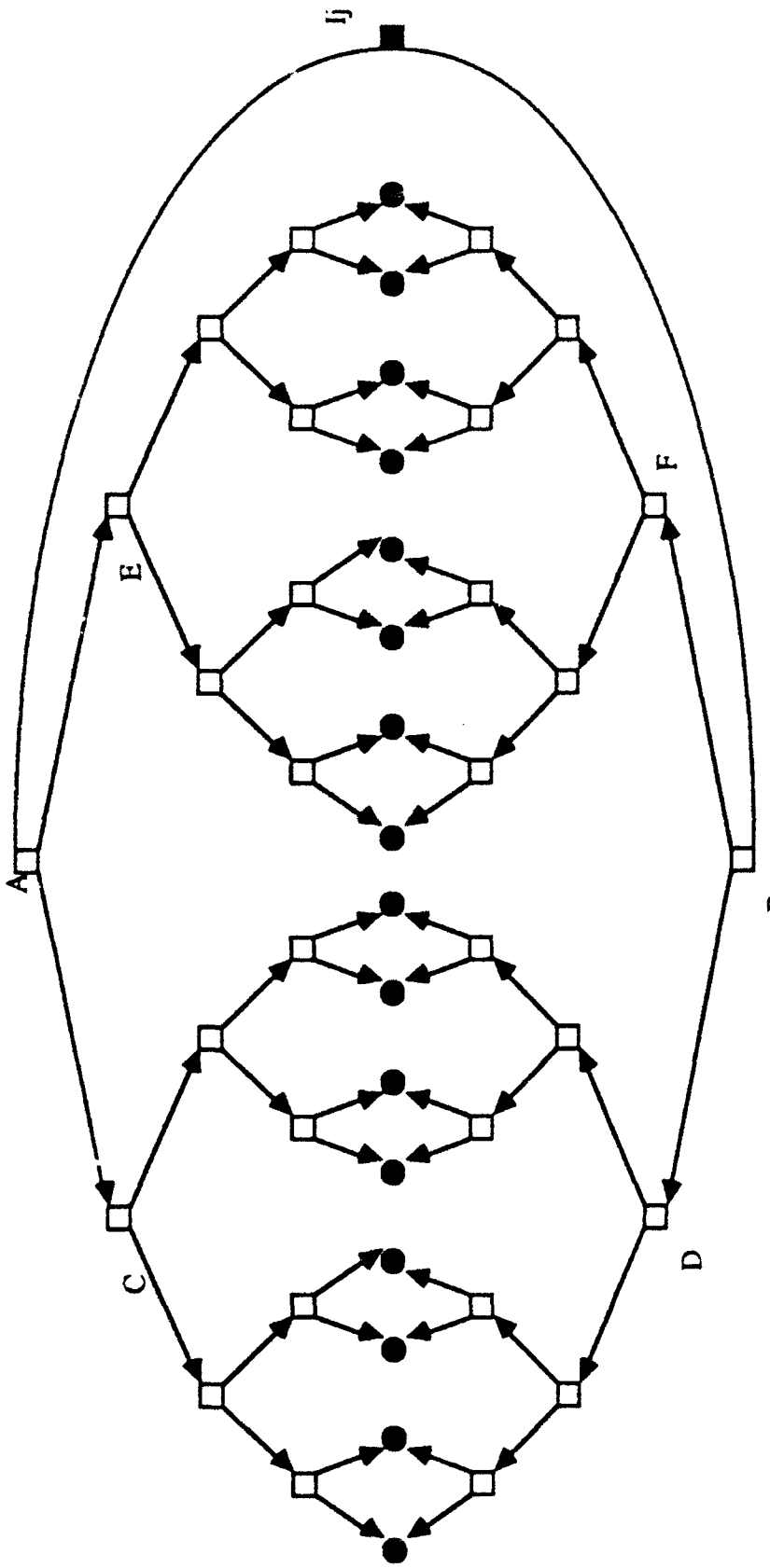


Figure 3. Paths from input Ij to all outputs